University of Alberta

CLOSED-LOOP INSULIN DELIVERY

by

Jamie Andrew Guay

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

in

Control Systems

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2001

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-60434-9

Canada

University of Alberta

Library Release Form

Name of Author: Jamie Andrew Guay

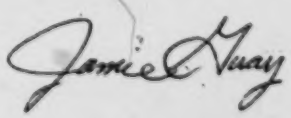
Title of Thesis: Closed-Loop Insulin Delivery

Degree: Master of Science

Year this Degree Granted: 2001

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis to lend or sell such copies for private, scholarly, or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



13212 – 62 Street
Edmonton, Alberta, Canada
T5A 0V6

March 29, 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Closed-Loop Insulin Delivery submitted by Jamie Andrew Guay in partial fulfillment of the requirements for the degree of Master of Science in Control Systems.

Horacio J. Marquez

Martin P. Mintchev

Rick E. Snyder

Qing Zhao

February 20, 2001.

Abstract

CLOSED-LOOP INSULIN DELIVERY

by Jamie Andrew Guay

Chairperson of the Supervisory Committee: Professor Horacio Marquez

Department of Electrical and Computer Engineering

The research undertaken during the course of this program was toward the development of a closed-loop insulin delivery system.

Using an implantable glucose sensor as the feedback element, an insulin fusion control system was created to continuously compensate for changing levels of glucose concentration in prospective diabetic patients.

The control system was tested *in vitro* and found to positively and consistently react to changing glucose concentrations when adequate glucose sensors were employed.

ACKNOWLEDGMENTS

Several individuals were instrumental in the initiation and development of this project. I would first like to give my sincere thanks to Dr. Horacio Marquez for his leadership. I would also like to thank Dr. Christie McDermott who provided me with a crash course in advanced chemistry. Her patience was greatly appreciated in what would prove to be the most challenging component of my research. I would also like to thank Dr. Jed Harrison for furnishing me with a place in his lab and for his helpful advice with regard to the sensor's testing.

I also express my sincere thanks to Dr. Ray Rajotte who provided the means for sensor testing and to Dr. Martin Mintchev for giving me the opportunity to work in what continues to be a vital and exciting field.

Special thanks also goes to my mother, who, as a dibètic for many years, was an inspirational encouragement. The support and encouragement of several friends during the course of my research was also greatly appreciated.

TABLE OF CONTENTS

Chapter 1 - INTRODUCTION 1
Current Treatment of Diabetes 2
Open-Loop Control 4
Closed-Loop Techniques 5
PC-Controlled Insulin Infusion 6
Competing Approaches 7
Chapter 2 - A CLOSED-LOOP CONTROL SYSTEM 10
Introduction 10
Implantable Glucose Sensors 13
Electrochemistry 16
Glucose Sensor Fabrication 17
Sensor Signal Detection 21
Connecting a Sensor 23
Analog-to-Digital Conversion 24
Sensor Calibration 26
Pump 30
The Controller 31
Software 31
Safety 32
User Interface 33
A Glucose Level to Infusion Rate Algorithm 45
Chapter 3 - GLUCOSE SENSOR TESTING 48
Introduction 48
Open Loop Testing 48
Results 49
Closed-Loop Testing 58
Chapter 4 - CONCLUSIONS 62
Insulin Infusion Control 62
Canine Testing 62
Future Research 63
Appendix A - ELECTRONIC DESIGN 65
A Pontentiostat 66
Voltage Amplification 74
Appendix B - SOFTWARE DESIGN 76
Design Approach 77

Code Structure 78
Serial Interfacing with the IVAC 570 80
Parallel Interfacing with the LabMaster 20009 84
Process Control 86
Error Protection 88
Main Module Program Listing 90

LIST OF TABLES

Table **Page**

Table 2-1. Desirable characteristics for an implantable glucose sensor..... 14

Table 2-2. *In Vitro* Characteristics of Needle-Type Glucose Sensor [17] 15

Table 2-3. Program command summary..... 38

Table 2-4 The sliding scale for determining the insulin infusion rate for a given blood-glucose concentration. This information provides the basis for a computer alogorithm..... 45

Table 2-5 Exceptional conditions to the insulin infusion rate algorithm..... 46

Table B-1 Summary of the functions contained within the program modules for the controller software..... 79

LIST OF FIGURES

Figure **Page**

Figure 1-1. Current approach to management of IDDM.....3

Figure 2-1. Generalized block diagram of a closed-loop insulin infusion system.....11

Figure 2-2. Block diagram showing each major component of the closed-loop insulin delivery system.....12

Figure 2-3. These graphs depict the body's natural and typical concentrations of blood glucose and corresponding insulin production.....12

Figure 2-4 A schematic diagram of a glucose sensor showing the Nafion-coated, working and reference electrodes.....19

Figure 2-5 A SEM micrograph of a glucose sensor.....20

Figure 2-6. A generalized block diagram of the glucose meter.....21

Figure 2-7. Front and side views of the glucose meter/potentiostat.....23

Figure 2-8. Sensor connection to glucose meter using modified jumper to produce a solderless connection.....24

Figure 2-9. (a) *In vitro* calibration scheme. (b) *In vivo* calibration based on the *in vitro* and *in vivo* values of the calibration parameters.....27

Figure 2-10. The IVAC 570 Variable Pressure Infusion Pump.....31

Figure 2-11. Screen capture of controller software prompting user for setup information.....34

Figure 2-12. Captured screen of the controller software showing updated blood glucose and pump infusion rates in real-time.....37

Figure 2-13. Captured screen of the controller software indicating total hourly amounts of insulin infused as well as the average infusion rate for the patient.....40

Figure 2-14. General program flowchart for closed-loop control.....42

Figure 2-15. Flowchart representing the CIM unit's validity checking algorithm for serial communications with the variable pressure pump.....44

Figure 3-1. Glucose meter voltage readings for known concentrations of glucose in 37 °C thermostated PBS, pH 7.4. A consistent correlation in sensor response was achievable from one measurement cycle to the next. $n = 5$.....51

Figure 3-2. Glucose meter voltage readings for known concentrations of glucose in 37 °C thermostated PBS, pH 7.4. Inconsistent and

fluctuating response due to imprecise control over fabrication and layer thickness. $n = 5$ 52
Figure 3-3. Average responses from 12 sensors. Each sensor displayed different linear characteristics. 53
Figure 3-4. Average responses for a sensor with readings taken two weeks apart. (The lower curve shows the latter response.) 55
Figure 3-5. Typical *in vitro* response characteristic for a sensor in five known glucose concentrations. 57
Figure 3-6. Typical closed-loop system response to a rapid increase in glucose concentration introduced to the system after 150 seconds. 59
Figure 3-7. Typical closed-loop system response when two rapid increases in glucose concentration occur in succession 61
Figure A-1. A circuit for controlling the potential at point *A* independently of the changes in the resistances R1 and R2. [12] 66
Figure A-2. An electrochemical cell model showing the electrode terminals 67
Figure A-3. A simple potentiostat circuit for controlling the potential at point *A* independently of changes in the sensor's impedance Z_S 69
Figure A-4. A potentiostat with a zener referenced source and feedback, which can be adjusted to produce the desired output voltage. 71
Figure A-5. Schematic diagram of the completed potentiostat/meter/amplifier. 75
Figure B-1 The process threads that execute in different combinations depending on the state of external hardware or user interaction. 87

Chapter 1

INTRODUCTION

It has been reported that diabetes is the seventh leading cause of death in Canada, affecting 2.6% of the population [1]. The seriousness of this disease has dramatically spurred on research in the areas of its treatment and cure. While there is yet no procedure that could be called a cure to diabetes, approaches to the treatment of diabetes, particularly insulin-dependent diabetes mellitus (IDDM) or Type I diabetes, have changed radically during the past ten years.

The American National Institutes of Health Diabetic Control and Complications Trial (DCCT) demonstrated that careful regulation of blood glucose levels results in significant reduction of long-term negative affects of diabetes mellitus [2]. A control scheme, which could deal with changing blood-glucose levels dynamically, would be of great benefit in the treatment of diabetes, particularly where circumstances warrant close and frequent monitoring.

This kind of monitoring would require some sort of biosensor that could continuously detect blood glucose concentration in the body. It would also be desirable to automatically compensate for continuously fluctuating levels of glucose in the blood by delivering the necessary amount of insulin in direct

response. This approach, in effect, would simulate the body's natural function for handling hyperglycemic or hypoglycemic situations, thus forming the basis for an artificial pancreas. Such a system could allow glucose levels to be more closely tracked and treated, preventing potentially dangerous levels occurring in the blood, and possibly promising an easier management approach for diabetics. The long-term benefits are also conducive to this approach, minimizing the adverse affects of diabetes patients often experience later in life.

This chapter continues with a summary of current treatment methods and proposed open- and closed-loop methods that have been employed. Chapter 2 contains descriptions of the main elements employed in constructing a computer controlled, closed-loop insulin delivery system using a needle-type sensor developed at the University of Alberta. Chapter 3 shows and explains the results of open- and closed-loop testing using the completed system, while Chapter 4 contains the conclusions and a summary of related research you can expect to see in the future.

Current Treatment of Diabetes

Insulin-Dependent Diabetes Mellitus (IDDM), or Type I diabetes (also sometimes refered to as Juvenile Diabetes because its symptoms often surface in adolescence.), is a disorder of the carbohydrate metabolism in which sugars in the body are not oxidized to produce energy due to a lack of the pancreatic hormone insulin. As a result, these sugars (glucose in particular) accumulate in the blood (hyperglycemia), then in the urine; symptoms include thirst, loss of weight, and the excessive production of

urine [3]. The use of fats as an alternate source of energy leads to disturbances of the acid-base balance, and the resultant accumulation of ketones in the bloodstream (ketosis), and can eventually lead to convulsions preceding a diabetic coma. Diabetes that begins in childhood or adolescence is usually more severe than that beginning in middle or old age.

Treatment of type I diabetes is based on a carefully controlled diet and is commonly managed with regular injections of insulin. The most widely accepted approach to the management of IDDM involves the process depicted in the block diagram of Figure 1.1. The regimental process of blood letting, such as pricking the finger, is followed by the actual glucose determination using any number of external glucose monitors that are commercially available. The BG (blood-glucose) concentration may also be determined manually, that is, by visual inspection of a test strip. More and more commonly, it is being reported by electronic means using a reflectance-type glucose meter. After devices like this are used, the need for an injection is determined and, if needed, a proper dosage is prescribed and administered by the individual [4].

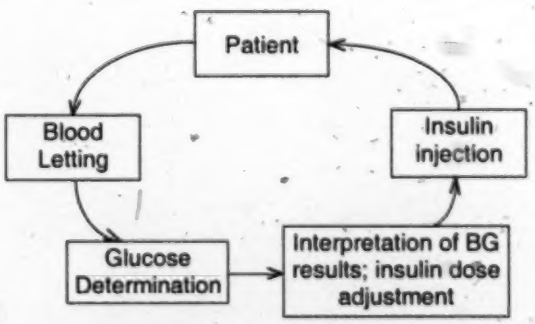


Figure 1-1. Current approach to management of IDDM.

Due to the nature of the disease and the dependence of blood glucose levels on food intake and stress, IDDM must be closely monitored, requiring blood sugar testing and charting at least four times a day [5].

The urgency in the development of a suitable method of treatment for IDDM has been influenced mainly by the seriousness of its effects on the body. Multiple complications can arise from this metabolic disorder, several of which appear as a diabetic gets older. Consequently, it is recommended that optimal regulation of glucose levels should be achieved in the treatment of diabetics who are at greater risk of the microvascular complications, primarily in young and middle aged persons [6]. The ideal treatment method then, would involve continuous monitoring and the compensatory management of elevated gluocose levels in the blood with the atomatic infusion of insulin, such as would be provided by a closed-loop system.

Open-Loop Control

Control systems which do not employ *feedback* are defined as open-loop control systems. In the context of the management of IDDM, such a system provides no means by which blood-glucose levels can be continuously monitored and compensated for in a manner that is consistent with the body's natural behavior. Feedback is the fundamental concept that separates open-loop control from closed-loop control. All of the current treatment methods already mentioned above, i.e., the use of test strips, and reflectance-type glucose meters, are thus open-loop control methods, since glucose levels cannot practically be determined on a continuous basis using these devices.

These approaches are rough estimation at best because they only provide the necessary information at a few discrete times of the day.

Significant advances, however, have been made in the area of open-loop control.

Closed-Loop Techniques

The key element to closing the loop is a reliable, robust, and minimally invasive blood-glucose sensor. The combination of such a device with additional components, such as a microcomputer to interpret the sensor's readings and make a determination of the necessary amount of insulin to patient required, as well as an insulin infusion pump to provide the actual delivery of insulin into the patient, would be fundamental to the system. These elements, in conjunction with the electronic infrastructure required to connect them together, would form the basis for a closed-loop insulin delivery system in which the patient received the proper dosage of insulin based on direct input from his own blood system.

A short term intravenous sensor similar to a catheter could be used in hospitalized patients for up to 3 days. This approach could be utilized to diagnose glycemic stabilization, manage ketoacidosis, or monitor blood-glucose levels during surgery or recovery. Other applications include monitoring glucose instabilities in mothers during delivery, certain neonates in intensive care, and other similar situations.

It is also quite plausable to insert a sensor subcutaneously over the short term in hospitalized or nonhospitalized patients for periods of several days. Such

an application could be connected to an external data storage and display device which could be a portable computer with a display screen next to the bed of a hospitalized patient. For nonhospitalized patients, such a device could take on the form of a compact belt-mounted unit, or possibly a wristwatch sized device.

PC-Controlled Insulin Infusion

This project addresses the need for a closed-loop insulin delivery system and demonstrates how a personal computer can act as the controlling mechanism for treating diabetic patients undergoing an islet transplant procedure, a surgical procedure, or delivery. These three medical situations form the basis for the necessary algorithms which must be in place to determine appropriate insulin infusion rates. The system employes a custom software design that incorporates computer interfacing for an Analog-to-Digital (A/D) converter as well as an IVAC infusion pump equipped with a Computer Interface Module (CIM). These connections are necessary so that the unique Nafion coated, needle-type glucose sensor, which was developed at the University of Alberta, could be connected to a custom-built, portable potentiostat and meter. This same device provided the necessary signal conditioning in order to supply a sensitive enough voltage to the A/D converter, which in turn provided a digitized value that is directly proportional to the glucose concentration it measured. The sensor would serve as the feedback element so that an accurate glucose level could be collected and a corresponding insulin infusion rate determined. Using the computer's interface to the CIM equipped pump, elevated levels of glucose could be nominalized by infusing insulin into the patient at the appropriate rate. Since the process of sampling

glucose and compensating for hyperglycemic levels with insulin is continuous, the system behaves in a manner much like the body's own pancreas would.

Competing Approaches

While the idea of controlling insulin infusion in a closed-loop is not new the methods have been few and varied. The idea of electronically assisted managment of diabetes has been revisited more frequently in the recent past because of advances in miniaturization and computer technology. However most methods have implements an open-loop method. In 1974, Slama *et al.* showed that insulin delivered from a portable infusion pump could significantly improve glycaemic control [7]. Since no suitable glucose sensors has been developed to be reliable enough to withstand the physiology of the human blood system and serve as the feedback element over the long term, the approach to automating the treatment IDDM patients has remained open-loop.

The Penject (Hypoguard) is a low-technology solution to insulin delivery and is designed for greater convenience and accuracy compared to convention injections. Taking the form of a pen, it attaches to a 1 ml insulin syringe, which the patient fills. A selection ring is rotated on the device to set the desired number of units to infuse. It can store enough insulin for up to three days of use and the user administers it when needed [8].

A more sophisticated device is the Pen Infuser (Markwell Medical), which is worn by the patient continuously while delivering boluses of insulin from a 3

ml syringe into a subcutaneously implanted cannula. A knob on the end adjusts a leadscrew which is connected to the syringe plunger. This device is designed for the diabetic that requires an intesified regimin where several infusions would be required throughout the day. At betime, however, the short-acting insulin runs out and separate injections of long-acting insulin must be used to prevent morning hyperglycaemia [9].

The compact design of the motor driven Nordisk Infuser (Nordisk, Gentofte, Denmark) comes prefilled with insulin for up to 10 days use. Instead of a motor driven leadscrew, which other bulkier devices have used, the motor drives a geared pinion attached directly to the syringe plunger shaft. The device is battery powered and can be worn portably [10].

Both Siemens Promedos and Windsor Medical have prouced commercial insulin infusers employing peristaltic pumps, whereby the rotary action of rollers moving along a flexible tube. The Siemens device can accommodate up to 30 ml of insulin offering a potential one month usage period of continuous operation. Its performance has been shown to be useful for long-term intravenous cannulation use [11].

Yet other approaches have used surgically implanted insulin infusion pumps that can be programmed remotely and refilled with a minute catheter [12, 13, 14]. The disadvantage lies in the invasivness of the implantation procedure. Some close-loop experiments have also been done using the an artificial endocrine pancreas (AP) (STG-11A, Nikkiso Co., Ltd., Japan). With this device, small amounts of venous blood is extracted from the body on a continuous basis, diluted, and checked with a glucose oxidase membrane.

The glucose is memorized and calculated, and insulin is then pumped into the body automatically. It was also shown to control hyperglycemia in diabetic patients [15].

Even with a variety choices for the treatment to IDDM, there is still new ground to be broken in what methods are best suited to a particular case. The project undertaken for this study was unique in its flexability and serves as an excellent test bed for current or emerging implantable sensor designs. The modular design of both the software and the hardware elemements of the system means that one component can be easily exchanged for another for research and experimental purposes. The closed-loop insulin delivery system constructed for this research used a unique, Nafion coated, needle-type sensor (developed at the University of Alberta), and demonstrated linearity and sensitivity. However, the system could serve as a host to other devices, including sensors, for experimentation. This is possible since no particular hardware component is dedicated to the overall system. Likewise, the software's modular design, means that any hardware component that does require a unique driver, the new code for it could be easily linked into the main application.

Another advantage to the approach used in this experiment is the possibility of networking multiple IVAC insulin infusion pumps. together Several patients could then be monitored from a central control area. Such a system could be used in a hospital where a limited number of staff monitor the blood glucose levels of multiple patients simultaneously.

Chapter 2

A CLOSED-LOOP CONTROL SYSTEM

Introduction

Automatic control involves machines only, in other words, human intervention would not be a part of normal operation. As with a room-temperature control, where the temperature is controlled by a furnace, it can be controlled (turned on and off) according to a automatic thermostat reading. The desired temperature is predetermined by the user and the thermostat issues the appropriate signal to the furnace to maintain that temperature. Thus, in a more general sense, control can be thought of as the process of causing a system variable to conform to some desired value, called a reference or set-point value [16].

A *closed-loop* control system is one in which *feedback* is used to provide the controller with an up-to-date report of the system variable's magnitude so that a comparison can be made between that value and the reference value. Once the comparison is made, the controller can then adjust its output as needed. In this experiment, the reference is the range for normal glucose levels in the blood system of a human being. A generalized depiction of the overall system is represented by the block-diagram in Figure 2-1.

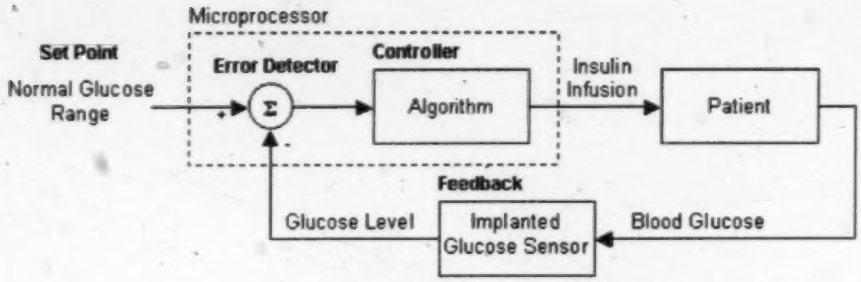


Figure 2-1. Generalized block diagram of a closed-loop insulin infusion system.

Error detection involves making the actual comparison between the known normal glucose value and the measured value determined from the sensor. An unhealthy discrepancy will cause the controller to invoke the necessary action based on an appropriate algorithm, that is to increase, maintain, lower or stop infusion of insulin into the patient.

A closed-loop insulin infusion control system must consist of two parts: a glucose sensitive biosensor and a microprocessor-based controller. The biosensor must be implantable and small enough to be readily replaced. For general application, the electronic controller must be small enough to be wearable in normal daily life. For clinical application and prototype analysis, the primary controlling components of this system are a desktop computer, a potentiostat, an analog-to-digital convertor, and a software controllable infusion pump. The functions of the controller are multiple: controlling the potential of the biosensor's electrode, sampling, filtering, storing and processing the current, and transforming the current generated by the sensor into an estimation of a glucose concentration.

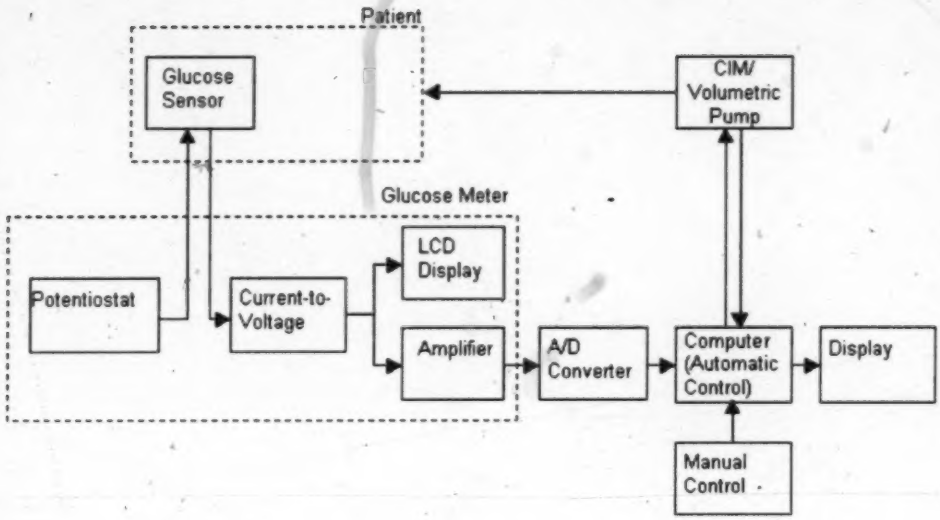


Figure 2-2. Block diagram showing each major component of the closed-loop insulin delivery system.

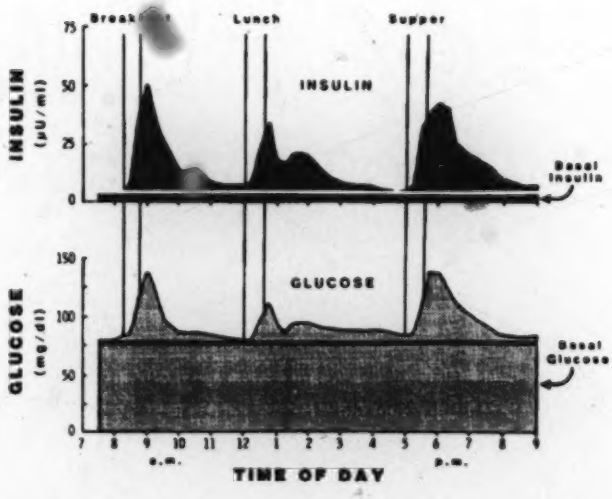


Figure 2-3. These graphs depict the body's natural and typical concentrations

of blood glucose and corresponding insulin production.

Implantable Glucose Sensors

Long-term maintenance of blood glucose at normal levels with a closed loop control system is one of the ideals of diabetic treatment. For this purpose, a continuous glucose-monitoring device is needed with accuracy of measurement, high specificity to glucose, quick response, and negligible blood loss as well as long life [17]. It is this element, a glucose sensitive biosensor, that separates the proposed system from those that are commercially available. The biosensor provides the necessary feedback to the system's controller so that a real-time, 'intelligent' decision can be made based on the patient's current condition, that is, in real time.

The purpose of any sensor is to convert a physical measurand (the quantity or property being measured) to an electric output. A glucose sensor must ideally be able to accurately detect the concentration of glucose (the measurand) in the blood. Unfortunately, other factors are present which can interfer and affect the outcome, such as ascorbate, uric acid, and acetaminophen [18]. However, the choice of dialysis membranes used can greatly enhance selectivity, to thus screen out unwanted factors while allowing the highest possible degree of permeation for glucose.

Several essential problems must be addressed in regards to the development and practical use of implantable blood-glucose sensors. Accuracy, of course, is one of the most important considerations. Obtaining accurate glucose

level readings have been achieved already with a number of different implantable glucose sensors [9]. Thus far, the greatest challenge to be met is with the sensor's longevity [19]. A bio-compatible sensor, which can function reliably over a long period in the harsh environment of the body, is desired if any provision for a better lifestyle in the diabetic patient is to occur. Sensor lifetimes may only number in days, limiting them to in-patient use, during surgical procedures or special monitoring times such as delivery, or islet transplant. Table 2-1 summarizes the desirable characteristics for an implantable glucose sensor for stable long-term use [20].

Table 2-1. Desirable characteristics for an implantable glucose sensor.

Operational Demands	**Range:** 20-2000mg/dl or 1-100 mM if undiluted blood is to be measured (or lower by a factor < 1 if measurement is performed in the tissue).
	Response time: < 10 minutes, the sensor does not need to be very much faster than the rate of appearance of the glucose in the blood
	Accuracy • Specific for glucose only (error due to interfering compounds < 10%) • Low base-line drift since direct zeroing is impossible after implantation • Low sensitivity drift. The slope of the calibration curve should be sufficiently stable not to increase the error above 10%. Weekly calibration should be sufficient (daily calibration would negate the benefit of the lack of a daily injection). • Low temperature dependence of the signal of about 5%/°C
Implantable	**Small size** and a shape that will not harm or impede

Demands	body fluids or incur long term side effects. **A one-year lifetime** that can be guaranteed so as not to be burdensome to the user. **Power consumption related to lifetime**. Battery recharging should be non-invasive. **Comfort:** The encapsulated and sterile sensor system should be free of reagent addition and exchanges of material in order to eliminate the risk of inflammation.

It has been shown that the subcutaneous glucose concentration detected by a sensor closely follows that of blood glucose with a 5-minute time lag [21, 22].

A unique needle-type glucose sensor employing a Nafion coating has been extensively investigated both *in vitro* and *vivo* in rats [23, 24], dogs [14, 25], and human volunteers [26]. In all cases, it was shown that the sensor current closely followed the plasma glucose levels. In the *in vivo* canine experiments a delay of 3 minutes was recorded for a peak current to be registered, which is the expected lag time for a blood glucose concentration to translate to subcutaneous tissue **[Error! Bookmark not defined.]**. Table 2-2 shows average sensitivities and background currents for the sensor employed for open and closed-loop control system analysis.

Table 2-2. *In Vitro* Characteristics of Needle-Type Glucose Sensor **[Error! Bookmark not defined.]**

Background current (nA)	20 ± 7 (n=19)
Sensitivity (nA/mM)	25 ± 10 (n=19)
Response time (s)	33 ± 13 (n=6)

The implantable glucose sensor developed by Moussey *et al.*, has been shown to provide superior glucose selectivity, durability and rapid response time compared to many other sensor designs. A shorter stabilization period following electrode polarization provides an additional advantage [9]. The sensor itself employs layers of poly(o-phenylenediamine, or PPD) undercoating with Nafion overcoating. The resultant combination provides improved performance in a biological matrix compared to the use of either layer by itself. For these reasons it is this sensor configuration that was chosen to serve as the feedback element.

Electrochemistry

Electroenzymtatic sensors depend on the enzymatic oxidation of glucose by oxygen (O_2) to form gluconic acid and hydrogen peroxide (H_2O_2) in the presence of the enzyme, glucoseoxidase (GOx). The chemical reaction is represented by

$$\text{Glucose} + O_2 + H_2O \xrightarrow{GOx} \text{Gluconic Acid} + H_2O_2 \quad (2\text{-}1)$$

This reaction is important because it allows the detection of the consumption of O_2 or the generation of either of the two substrates produced: gluconic acid and hydrogen peroxide. The sensor is not directly sensitive to glucose, so the detection of any of the other three molecules will provide an indirect means of determining blood-glucose concentration. The most commonly used and simplest detection method to implement is that of hydrogen peroxide production [27,28,29,30]. Hydrogen peroxide can be measured

electochemically at a potential of about +0.7 volts versus saturated calomel electrode (SCE), as shown in equation (2.2) below.

$$H_2O_2 \xrightarrow{+0.7\ V} O_2 + 2H^+ + 2e^- \quad (2\text{-}2)$$

Glucose Sensor Fabrication

The supporting component of the needle-type sensor consisted of a varnished copper wire (approximately 10 cm in length) with 0.5 mm diameter. One milimeter of varnish was removed from one end of the wire to allow electrical contact with a platinum wire which was coiled 3 times around the stripped region of the copper wire and then an additional 7 times farther along the wire. A silver wire 0.1 mm in diameter was coiled 15 times around the copper wire 1 mm from the platinum coil. The trailing end of the silver wire was connected to another varnished copper wire 0.1 mm in diameter that was also stripped 1 mm at one end. The sensor was then washed in 0.1 M HCl solution for 5 minutes and rinsed in double-distilled water. Regions of exposed copper, where electrical connections were made, were then insulated using insulating varnish. The Pt-Cu connection was given two coats of green GLTP insulating varnish and a final coat of red varnish. The Ag-Cu connection was given two coats of Red varnish only. The sensor was dried in air overnight. It was small enough to fit through a 20 gauge catheter.

The exposed remainder of the coiled platinum wire was anodized for 5 minutes at 1.9 V versus SCE in 0.5 M H_2SO_4 to depurate the surface. The platinum wire was then cycled between –0.26 and +1.1 V versus SCE for 5 minutes. During this procedure, a cyclic voltamigram (CV) plot was

produced to confirm a Pt signature curve and allow the cleanliness and connections to be checked. Silver chloride was then formed on the silver wire by anodizing potentiostatically at +0.08 mA for 30 minutes in stirred 0.1 M HCl.

A small quantity of enzyme solution was made by first preparing 1 mL of glutaraldehyde acetate buffered solution (5 mg/mL of glutaraldehyde) and slowly dissolving 73.2 mg of bovine serum albumin (BSA) into this stirred buffer solution. Then, 200 mL of this solution was added to a 3.9 mg aliquot of glucose oxidase (GOx). The enzyme was deposited on the sensor's working electrode (Pt coil) by passing it through the loop formed at the end of a short length of wire that was previously dipped in the final enzyme solution.

The sensor was air dried for one hour at room temperature. The sensor end (both Pt and Ag/AgCl coils) was then sequentially dip coated with 0.5 wt%, 3 wt%, and four layers of 5 wt% Nafion. The sensor was allowed to dry for 30 minutes between each Nafion coating and was finally air-dried overnight.

The sensor's Pt electrode was electropolymerized with 1,3 phenylenediamine (1,3 DAB) by adding 5 mM of 1,3 DAB to acetate-buffered (pH 5.5) solution and anodizing at +0.65 V versus SCE. The solution was degassed and blanched by nitrogen. After being carefully rinsed in D.D. water and allowed to dry, the sensor was heat sterilized in an oven at 120 °C for 120 minutes.

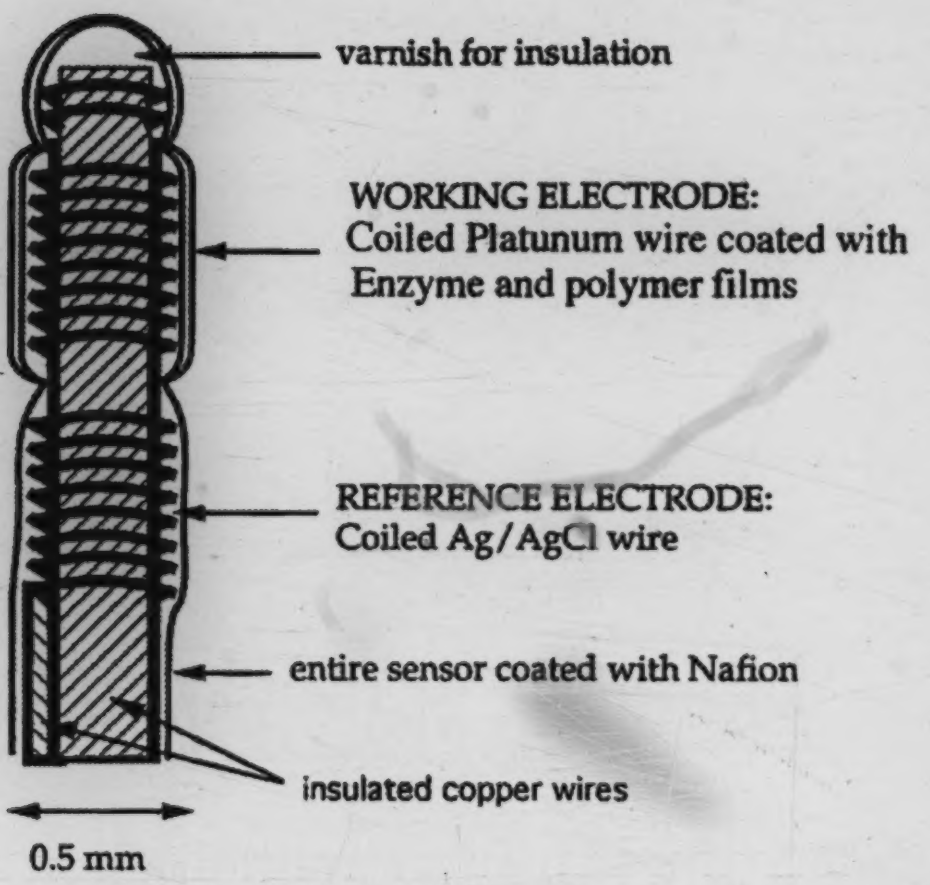


Figure 2-4 A schematic diagram of a glucose sensor showing the Nafion-coated, working and reference electrodes.

Figure 2-5 A SEM micrograph of a glucose sensor.

Sensor Signal Detection

A glucose meter was specially designed for the needle-type glucose sensor to act as a combined potentiostat, signal amplifier and monitoring device. It is this device that must ultimately interface the implanted glucose sensor in the host subject to the computer. It must reliably and consistently supply the necessary potential for the sensor and operate and report the occuring signal from the sensor to the controller. A block diagram for this device is shown below. Blocks outside the dashed box are externally connected to the glucose meter with appropriate lead wires.

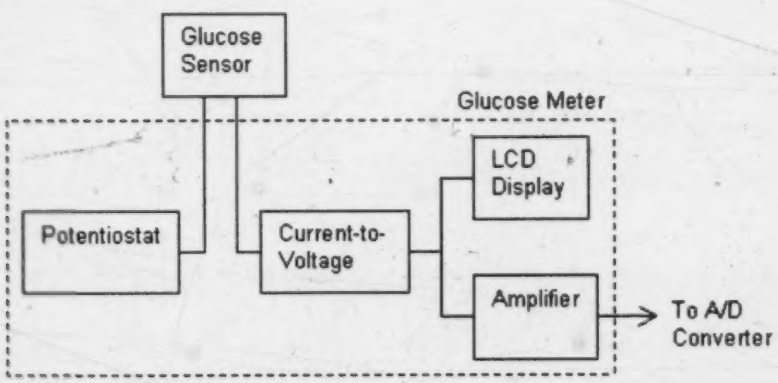


Figure 2-6. A generalized block diagram of the glucose meter.

In order for the sensor to function, a voltage of 0.7 volts is required across the sensor's leads [5]. This potential supplies the necessary catalyst for producing a measurable current by the separation of hydrogen ions from oxygen from the available source of hydrogen peroxide in the blood (see Equations 1 and 2). This is handled by a potentiostat. The measure current is

transformed to a voltage and displayed on a liquid crystal display built into the same device. The voltage is also amplified for connection to the LabMaster's Analog-to-Digital convertor for subsequent reading by the computer's controlling software.

The device uses two 9-volt batteries, one for the sensor circuitry and one for the 3-1/2 digit LCD display (Novatron, N-128), since the voltmeter requires an independant supply. A push button toggle switch turns the LCD display on or off but does not affect the sensor circuitry. The sensor circuitry is on as long as the battery is connected inside the battery compartment. The meter's hardware is contained inside a plastic PACTEC® box (K HML-ET-9VB-000). Figure 2-9 shows front and side profile view of the glucose meter device.

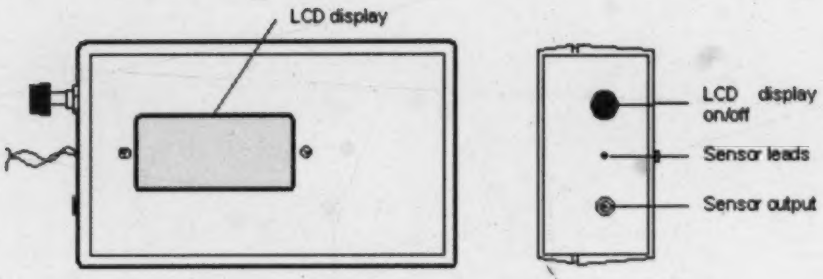


Figure 2-7. Front and side views of the glucose meter/potentiostat.

A mini-phono jack is available so that the meter may be connected to a chart recorder or A/D converter that in turn is connected to a PC (specially constructed cables allow for this type of connection). The voltage at this output may range from 0 to approximately 3 volts.

The twisted pair of sensor leads will connect to a glucose sensor using the method described in the following section. The bias voltage may be calibrated by adjusting the trimmer potentiometer using a small screwdriver. To calibrate the sensor's bias voltage, attach a voltmeter to the sensor leads of the meter, then insert a small screwdriver in the hole on the back of the meter box. Adjust the trimmer pot until the voltmeter displays 0.7 volts. The unit can also be equipped with a belt clip to facilitate portable use by the patient.

Connecting a Sensor

A needle-type glucose sensor may be connected to the meter by first removing the modified circuit-board jumper from the two-pin connector at

the end of the long pair of leads coming from the meter. The jumper was altered by removing the metal bridge, so that it does not short the two sensor electrode together. Excess varnish and oxidation were scraped from the ends of the sensor electrode using a small knife. The cleaned leads are then inserted into the holes of the modified jumper, and it is then reconnected to the two-pin connector, establishing a snug solderless contact.

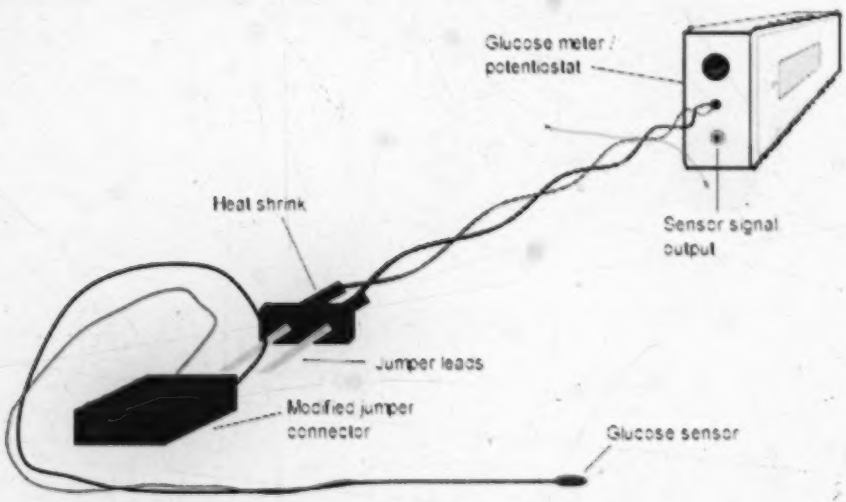


Figure 2-8. Sensor connection to glucose meter using modified jumper to produce a solderless connection.

Analog-to-Digital Conversion

Proper conditioning of the sensor's analog signal is needed before is can be analyzed by the computer. This conditioning takes the form of analog-to-digital (A/D) conversion. An analog, or continuous signal, when sampled yields a signal related magnitude independent of the time at which it is

sampled. A digital, or discrete signal, on the other hand, is one which reflects the signals magnitude only at regular instances in time. Thus A/D conversion involves taking periodic samples of an analog signal and assigning discrete values at those moments which a digital controller or computer in turn can receive as a binary value [31].

A LabMaster combination A/D-D/A convertor (model 20009, Scientific Solutions Inc.) was used to interface the sensor meter/potentiostat to the computer. The physical device is composed of a mother board that internally plugs into an available PC slot. The mother board is then cabled externally to the LabMaster containing the daughter board, which performs the actual conversion from analog data to digital data which in turn the computer can use. The LabMaster is equipped with 16 separate input channels for A/D conversion, each with its own addressable I/O port for computer access. Four channels were simultaneously used to initially test interfacing and signal detection. Since the A/D circuit inside the LabMaster can only convert one analog voltage to its digital equivalent at a time the LabMaster's internal multiplexing capabilities make it possible for each channel (designated 0 through 15) to be strobed for a value in succession. Thus the four channels are each read, one at a time, then the cycle is continuously repeated by strobing the A/D through a software command.

The actual conversion process involves two steps. The analog multiplexer circuit of the LabMaster is told by the computer (software controller) which channel to switch along, the controler then samples it and holds that value in an analog memory cell (using a sample and hold circuit) until it is ready to be converted. In this control system, the end of the first step is designed to

trigger the second. The second converts the value to a digital equivalent which the computer can then process.

The standard A/D convertor on the LabMaster daughter board itself has a maximum conversion rate of 30 kHz and user selectable input ranges of 0 to +10 volts and -10 to +10 volts.

Sensor Calibration

The calibration of the glucose sensor is a necessary and critical process for the subcutanesous implantation of such devices, since the characteristic behaviour of one sensor compared to another may differ. As a result, the calibration process allows for an accurate estimation of the prevailing subcutaneous glucose concentration from the sensor's output. Calibration is based on the comparison of two points of glucose concentration with their concomitant points of sensor output. From these two values, it is possible to calculate a sensitivity coefficient for the sensor, expressed as the ratio between the changes in the sensor output and those in glucose concentration ($\Delta I/\Delta C$), and the background current (converted to and measured as a voltage) that corresponds to 0 mol/l glucose (Figure 2-9(a)).

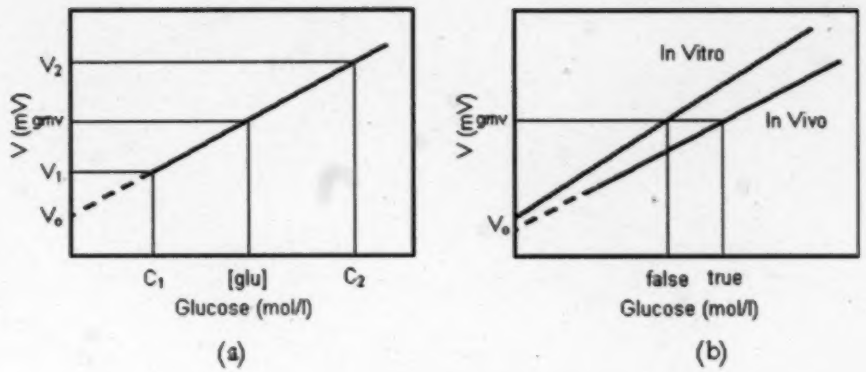


Figure 2-9. (a) *In vitro* calibration scheme. (b) *In vivo* calibration based on the *in vitro* and *in vivo* values of the calibration parameters.

Velho *et al.* have listed three strategies for calibrating a subcutaneously implanted glucose sensor: an *in vitro*, a one-point *in vitro*/one-point *in vivo*, and a two-point *in vivo* calibration procedure [32]. The *in vitro* calibration procedure involves placing the glucose sensor in a thermostated (35-37 °C), stirred, phosphate buffered saline (PBS) solution with a pH of 7.4 (to simulate the environment experienced by a subcutaneuouly implanted sensor as closely as possible), during stepwise increases of glucose concentration from 0 to 20 mM/l. The resulting sensor's linear characteristic can thus be expressed in mV/mM/l glucose and a calibration constant can be determined from this ratio, i.e. the slope of the resulting characteristic. This approach assumes that both the measured voltage in the absence of glucose (V_0) and the sensitivity of the sensor or identical under *in vivo* and *in vitro* conditions. While Velho *et al.* points out that the calibration could indeed be based on *in vitro* values (on which this experimental control system is based), the sensor's

response characteristic could be quite different under *in vivo* conditions. Applying the sensitivity coefficient and background current parameters obtained by *in vitro* calibration could result in a miscalculation of the subcutaneous glucose concentration (Figure 2-9(b)). The one-point *in vitro*/one-point *in vivo* calibration procedure makes the assumption that the values of V_0 measured in the absence of glucose are the same and relatively small for both *in vitro* and *in vivo* situations and thus only a single *in vivo* measurment at a know glucose concentration would be needed to define the sensor's characteristic. The assumptions made in these first two calibration procedures can be avoided by using a two-point *in vivo* calibration procedure in which two sensor voltage measurements are recorded at two different times when the blood glucose concentrations are known to be different, i.e. before and after an insulin injection or meal. Voltage readings made directly from the sensor are related to blood glucose concentrations determined through a conventional method, such as with a glucose analyzer.

While a two-point *in vivo* calibration would be required to best reflect the true sensor response for subcutaneous applications [32]. The best sensor calibration method for the *in vitro* collection of glucose concentration data and overall closed-loop control system analysis was chosen to be a two-point *in vitro* calibration procedure.

Using the slope equation a single value is obtained, uniquely describing the sensor's response characteristic as depicted in Equation (2-7). The meter gain, G_M, is included to account for the amplified signal supplied to the computer.

$$K_S = \frac{\Delta \text{Glucose Level (mM/L)}}{G_M \times \Delta \text{Meter Voltage (mV)}} \qquad (2\text{-}3)$$

The actual glucose concentration can now be calculated by interpolating the value using the sensor's known characteristic. Equation (2-8) describes the overall system equation for determining the actual blood glucose concentration based on the sensor's output.

$$BG = K_S \times \frac{D_{LM}}{R_{LM}} \times P_{LM} + C_2 - K_S \times G_M \times \frac{V_2}{1000} \qquad (2\text{-}4)$$

where BG is the blood glucose level, D_{LM} is the digital glucose value read from the LabMaster, K_s is the sensor's sensitivity constant, P_{LM} is the peak voltage level of the LabMaster, R_{LM} is the digital resolution of the A/D convertor, C_2 is the known concentration of glucose in which the upper voltage characteristic of the sensor is measured, G_M is the meter gain of 15.3, and V_2 is the average upper voltage characteristic of the sensor. The last two terms of Equation (2-8) indicate the y-intercept of the previous graph, V_o.

To demonstrate, an example calculation is shown here. The *in vitro* calibration was performed in a thermostated (37 °C) phosphate buffer solution (0.1 mol/l NaCl, 10 mmol/l Na_2HPO_4, pH 7.4) containing known concentrations of glucose. After repeated observations of a sensor's response in 3 mM/L and 20 mM/L glucose/PBS solutions, average corresponding voltages from the glucose meter were observed. Values of 12 mV and 156 mV were observed in the low and high concentrations of glucose solutions

respectively. From Equation (2-7) above, the constant describing the sensor characteristic is

$$K_S = \frac{(20-3)\text{mM/L}}{15.3} \times \frac{1000\text{mV/V}}{(156-12)\text{mV}} = 7.72\ \text{mM/L/V}$$

It is now possible to determine a corresponding glucose concentration within the sensor's detection range using Equation (2-8). To test its accuracy, the glucose sensor was placed into other glucose solutions of known concentration, then the computer's report of the concentration was observed and compared with the known value. When the sensor was placed in the solution of 15 mM/L glucose, an average value of 112 mV on the glucose meter was observed. This measurement when amplified by the meter, read and digitally converted by the LabMaster corresponds to a discrete value of 352. It is this value that the computer in turn receives.

$$BG = 7.72\text{mM/L/V} \times \frac{352}{2047} \times 10\text{V} + 20\text{mM/L} - 7.72\text{mM/L/V} \times 15.3 \times \frac{156\text{mV}}{1000\text{mV/V}}$$

$$BG = 14.85\ \text{mM/L}$$

Pump

The component responsible for the actual delivery of insulin must be a pump capable of digitally communicating with a typical personal computer. The IVAC 570 Variable Pressure Pump is equipped with a computer interface module or CIM allowing standard serial communications to occur between a PC and the pump.

Figure 2-10. The IVAC 570 Variable Pressure Infusion Pump

The Controller

The controller in this control system consisted of a 486 desktop computer. Except for the installed LabMaster 20009 daughterboard, the computer itself required no distinctive components. Two modes of interfacing were required: a standard serial link connected the computer to the CIM unit of the volumetric pump, and the LabMaster 20009 daughterboard was installed in an available PC slot, which in turn was connected to the LabMaster connector box containing the A/D converter. At the heart of the controller is, of course, the governing software.

Software

Software was custom made to handle timing, hardware interfacing, monitoring and display, and of course, the necessary algorithm to handle the determination of the appropriate insulin infusion rate based on the measured glucose level.

The software was developed in the C language, and will execute on any computer system running the MS DOS™ or Windows™ operating systems. The software was designed using modular procedure-oriented programming techniques and was developed and compiled with Borland®'s Turbo C++ integrated development environment, version 3.0. The software can also be modified and upgraded with additional modules to accommodate algorithmic changes or alternatives, different pumps, or to meet medical staff needs. The software is also independent of the type of sensor that is used; other sensor types could be easily substituted. Refer to Appendix B for a summary of the software development stage and a code listing of the primary control loop.

Safety

The most fundamental aspect in the design of a control system that must interact with a human body's function is that of safety. Software engineering practice defines the software component of this system as *primary, safety-critical software*, that is, software which is embedded as a controller in a system. As such, if the software malfunctions, the hardware can behave unpredictably and possibly result in human injury [33].

The current algorithm does, however, provide a suitable warning or alarm when a test subject's glucose level has significantly changed from one reading to the next. The subject or medical staff can then make a judgement as to whether the patient's glucose level has actually changed dramatically or if the sensor (or some other component) has failed.

Accidents are an inevitable part of complex system and software tends to increase the complexity of a system, so software control *may* increase the probability of a system malfunction. However, software controlled systems can increase the overall system saftey, since they can keep track of a wider range of conditions than electromechanical systems and they can be adapted relatively easily. The computer hardware they employ is known to have a high reliability and can also be designed to be portable. While software control may introduce additional complexity and possible hazards, the benefits in the hazards they can guard against often outweigh them. Appendix B outlines the features of the software and IVAC pump for dealing with error protection.

User Interface

The user interface design was done after consultation with doctors and nurses at the University of Alberta Hospital. This was to make certain that relevant information could be presented to medical staff in a clear way and with minimal operational complexity.

When the program is executed, the user is required to enter correct values for the date and time since accurate record keeping of a treatment session is required. The patient's name, weight, physician, and hospital number are then entered. The file name for a data file must also be specified. This file will store the patient information, measured glucose concentrations, and insulin infusion rates (Figure 2-11).

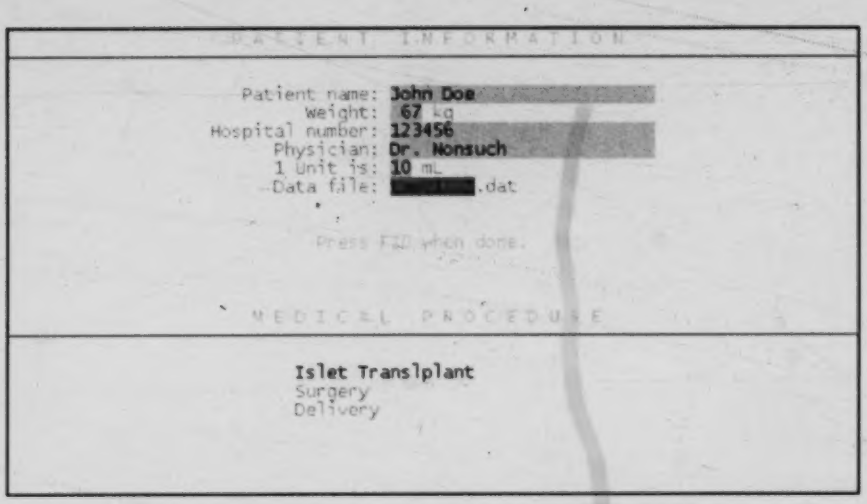


Figure 2-11. Screen capture of controller software prompting user for setup information.

Three separate algorithms determine how blood glucose levels will be handled depending on the medical procedure being implemented on the patient: Islet Transplant, Surgery, or Delivery. A menu is displayed to allow the user to select which procedure, and thus which glucose level-to-infusion rate algorithm, will be used. Once chosen, a new screen is displayed showing a menu allowing the user to choose one of three control modes: Manual Control, Infusion Control, and Sensor plus Infusion Control. The "Manual Control" option means that glucose concentration values will be directly inputed into the computer using the keyboard and will not be measured using the glucose sensor. This control mode will not send instructions to the variable pressure pump, but will instead report to the user what the pump setting should be manually set to. "Infusion Control" is similar to manual control in that glucose levels must be manually input, but instructions are

sent to the CIM unit of the pump via the computer's serial connection to cause it to infuse insulin at the necessary rate. The final control mode, "Sensor plus Infusion Control", samples signals from the glucose sensor via the LabMaster and also send instructions to the pump. This option is used for both the open-loop and closed-loop tests. If this control method is chosen, an additional screen displays a series of prompts so that a two-point calibration of the sensor can be performed using high and low measurements of a glucose solution of a known concentration.

The first two control modes represent open-loop control methods in which the insulin infusion would be carried out by medical staff. The third mode is a closed-loop method meaning that the software controls both the monitoring of blood glucose levels via the glucose sensor and the infusion of insulin via the external pump. Human intervention would ideally not be required when the software is running in the third operting mode, but may be exercised if necessary. These alternate operating modes offer medical staff a choice as to the level of control they wish to implement for a given situation. They also provide useful algorithmic and operational troubleshooting methods for design purposes.

The following figure (Figure 2-12) shows a condensed-time simulation of the treatment of a diabetic patient. In this example, blood glucose concentrations were entered manually and plotted on the left. Corresponding insulin infusion rates were calculated and displayed in the graph on the right. The screen offers a number of keyboard commands to the user that are displayed along the top in a menu bar. Since timing is important not only in data collection, but in the scheduling of required insulin injections, the current time is

displayed in the upper right-hand corner as well. The available keyboard commmands are summarized in Table 2-3. Pressing the key indicated in bold will initiate the command.

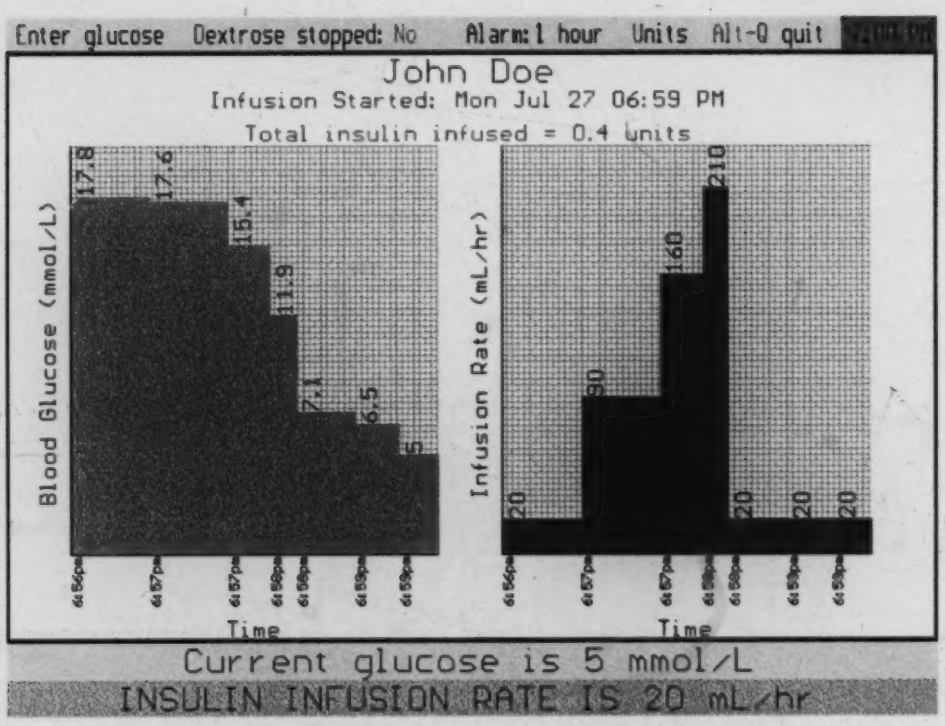


Figure 2-12. Captured screen of the controller software showing updated blood glucose and pump infusion rates in real-time.

Table 2-3. Program command summary.

Keyboard Command	Function
Enter Glucose	This command allows the system operator to manually enter a blood glucose concentration after a traditional glucose reading is taken from the patient. The insulin infusion rate/concentration is then determined. Pressing the ESC key will abort an entry. This command is available for manual system control or open-loop pump control.
Dextrose stopped/started	This command many be used for record keeping to indicate when dextrose is either being administered or has been stopped. The appropriate time and message is recorded in the patient's data file.
Alarm off/on [↑↓]	Activated using the up or down arrow keys, it controls a timer that will sound an audible alarm after a spedified amount of time.

Average	This displays the average insulin units that have been infused since the beginning of the procedure in the blue message area at the bottom of the screen. An additional graph is displayed showing the hourly insulin units delivered over the last 24 hours along with the average rate/concentration.
Unit	Intravenous units of insulin are defined as 10 mL by default, but these units may be re-defined by using this command.
Alt-Q	Quits from the program after confirmation. The command can be aborted by pressing N or ESC.

Pressing the 'A' key will display a screen like the one shown in Figure 2-13. This dynamic chart will indicate to the user the hourly amount of insulin infused into the patient in 'units' where a single 'unit' represents 10 mL of insulin by default. (Note that the 'unit' value may be explicitly defined during program setup.) The overall average amount of insulin infused is also shown as a solid line superimposed over the bar chart.

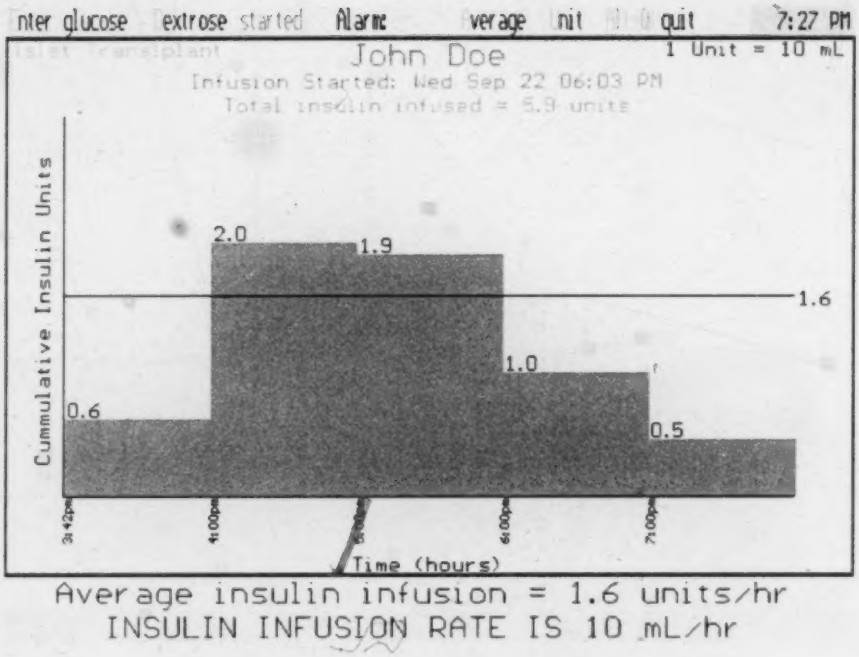


Figure 2-13. Captured screen of the controller software indicating total hourly amounts of insulin infused as well as the average infusion rate for the patient.

In the first operating mode (manual) blood glucose levels may be collected in the usual way, i.e., using an external glucose monitor. The glucose level may then be entered in mmol/L by pressing the 'E' key on the computer's keyboard. As soon as the value is entered, the appropriate insulin pumping rate is determined and both graphs are updated in real-time. Medical staff can then apply insulin as the specified rate by manual operation of the infusion pump. The 'infusion control' operating mode also requires that the patient's blood glucose level be determined and entered manually as in the manual

operating mode, but in this case the software directly controls the pump and automatically adjusts the infusion rate based on the entered glucose value. The third operating mode does not rely on the manual input of glucose, instead, medical staff may simply monitor a continual diabetic's treatment via the graphic interface. Blood glucose concentration is measured by the subcutaneously implanted sensor, which provided the continuous feedback signal to the software controller. Of course, should a problem occur a doctor or nurse could intervene at any time and revert to a different control mode.

The flowchart in Figure 2-14 outlines the general processes behind the execution of the computer software. All areas that require user input also include validity checking that is not necessarily shown.

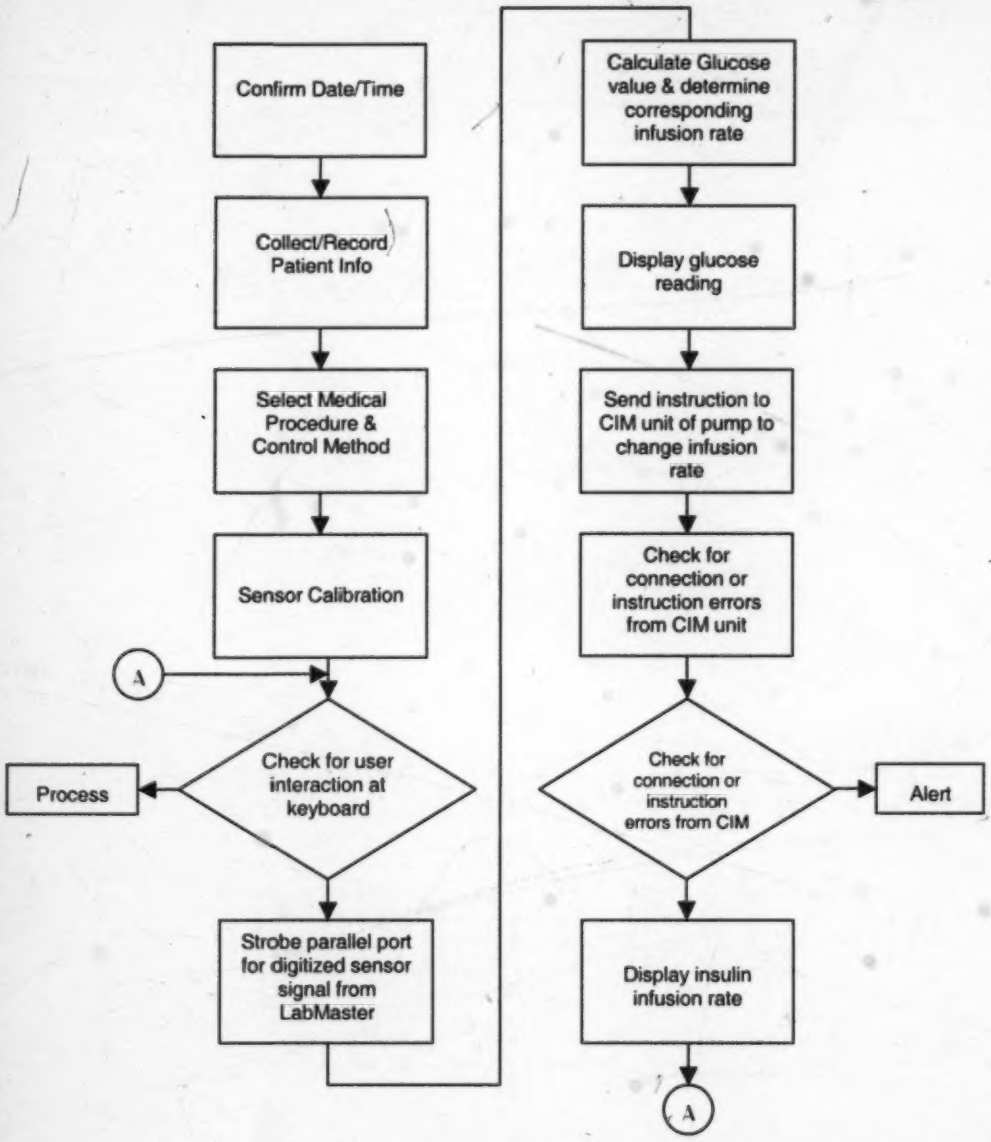

Confirm Date/Time
Collect/Record Patient Info
Select Medical Procedure & Control Method
Sensor Calibration
A
Check for user interaction at keyboard
Process
Strobe parallel port for digitized sensor signal from LabMaster
Calculate Glucose value & determine corresponding infusion rate
Display glucose reading
Send instruction to CIM unit of pump to change infusion rate
Check for connection or instruction errors from CIM unit
Check for connection or instruction errors from CIM
Alert
Display insulin infusion rate
A

Figure 2-14. General program flowchart for closed-loop control.

When any command is issued to the computer interface module (CIM), it must be subjected to various checks to make sure that it is in fact a valid commmand. Proper validity checks prevent the chance of any random series of characters, or the result of a communications error being misinterpreted as a CIM command.. The first three decision blocks in the CIM flowchart of Figure 2-15 represent these checks. If the command does not include an initial "U" character, it is simply ignored, otherwise the CIM proceeds to the next check. Subsequently, if a parity error, framing error, or overrun error occurs during the transmission of the command, the communications error is reported to the controlling computer and the command is reissued. If the actual command code is not one that is recognized by the CIM, the controller is also informed with an acknowledgment so indicating otherwise the valid command is determined to be one of either two types of commands: action or non-action. A non-action command is one which causes no change in the pump's current operation, but is designed to interrogate the pump for it's current status or preprogrammed settings. (Refer to Appendix B for examples of CIM command sytax.)

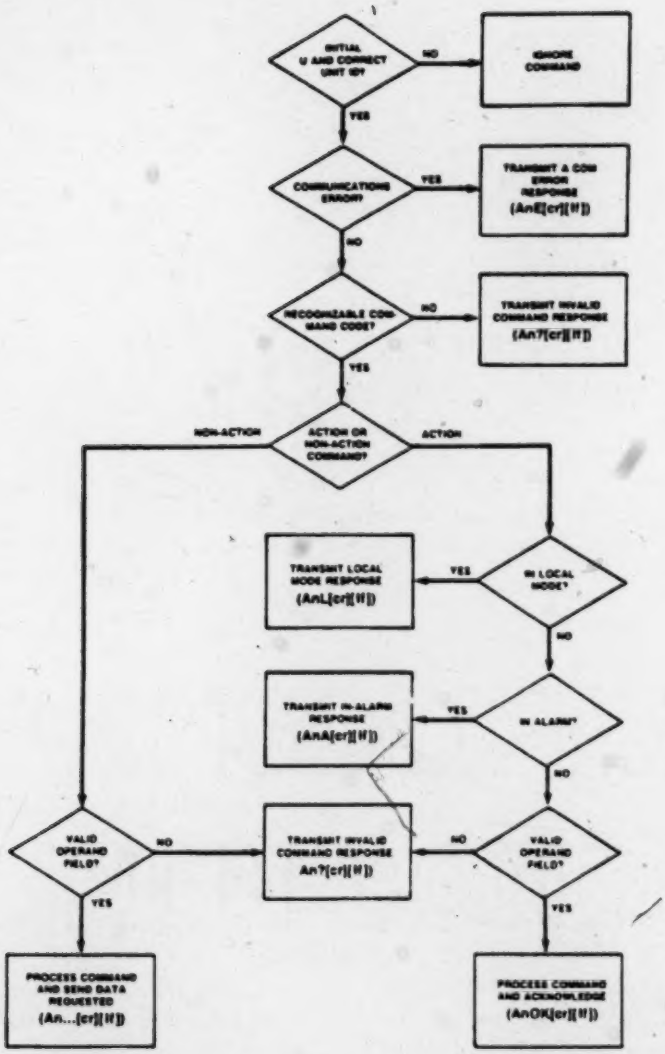


Figure 2-15. Flowchart representing the CIM unit's validity checking algorithm for serial communications with the variable pressure pump.

A Glucose Level to Infusion Rate Algorithm

Table 2-4 represents one algorithm that is implemented for diabetic patients in hospitals.

Table 2-4 The sliding scale for determining the insulin infusion rate for a given blood-glucose concentration. This information provides the basis for a computer alogorithm.

Initial setup:

1. Calories when the patient is NPO are provided in the form of $D_{10}W$, 100 ml/hr.
2. Insulin drip is made up as follows: 50 units of human regular diluted in 500 ml of normal saline, i.e. 1 unit = 10 ml.
3. The insulin is started at an infusion rate of 20 ml/hr.
4. The capillary blood glucose is checked hourly. After each I.V. dose of steroids and after each meal, the glucose is checked every 30 minutes for two hours.
5. The insulin infusion rate is altered according to the following schedule with each determination of the blood-glucose: ↓ means 'decrease' and ↑ means 'increase'

Measured Glucose Level (mmol/L)	Insulin Infusion Action
≤ 3	Stop insulin infusion
3.1 to 3.5	↓ to 10 ml/hr
3.6 to 4.0	↓ by 10 ml/hr
4.1 to 6.0	Remain at current rate
6.1 to 6.5	↑ by 5 ml/hr
6.6 to 7.0	↑ by 10 ml/hr
7.1 to 7.5	↑ by 15 ml/hr
7.6 to 8.0	↑ by 20 ml/hr
8.1 to 9.0	↑ by 30 ml/hr
9.1 to 10	↑ by 40 ml/hr
10.1 to 12	↑ by 50 ml/hr
12 to 14	↑ by 60 ml/hr
> 14	↑ by 70 ml/hr

This sliding scale was provided by medical staff at the University of Alberta hospital and is in current use there. By their request is was implemented to

accommodate an easier and more predictable transition to the into the software's use and behaviour. Table 2-5 extends the previous algorithm by accounting for situations when the glucose level increases rapidly.

Table 2-5 Exceptional conditions to the insulin infusion rate algorithm.

- If the infused glucose has been stopped for a period restart the insulin at 20 ml/hr.
- If the glucose drops ≥ 2 mmol/L between two consecutive readings to < 6 mmol/L, ↓ to 15 mL/hr.
- If the glucose drops ≥ 2 mmol/L between two consecutive readings to < 8 mmol/L, ↓ to 20 mL/hr.
- If the glucose drops ≥ 2 mmol/L between two consecutive readings to < 10 mmol/L, ↓ to 30 mL/hr.

All of the data entered and generated by the program is dynamically recorded into a text file while the program runs. The sample data below represents actual (abbreviated) data produced by a closed-loop session. This data can be easily extracted from the resulting file and imported from within a spreadsheet program to accommodate graphical and statistical analysis.

Patient name: Joe Smith
Weight: 67 kg
Hospital number: 123456
Physician: Dr. Nonsuch
Procedure: Islet Transplant
Date: Wed Sep 16 04:51 PM
1 Unit = 10 mL
Sensor number: 27
3 & 20 mM/L Meter Readings: 10.0 180.0
Meter gain: 15.3

	Glucose Measured	Blood Glucose	Insulin Infused	Insulin Infusion Rate	Total Units Infused
1	Wed Sep 16 04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
2	04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
3	04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
4	04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
5	04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
6	04:51 PM:	4.8 mmol/L	4:51pm:	15 mL/hr	0.0 units
7	04:51 PM:	4.7 mmol/L	4:51pm:	15 mL/hr	0.0 units
		...			
102	04:59 PM:	4.6 mmol/L	4:59pm:	15 mL/hr	0.2 units
103	04:59 PM:	4.9 mmol/L	4:59pm:	15 mL/hr	0.2 units
104	05:00 PM:	5.1 mmol/L	5:00pm:	15 mL/hr	0.2 units
105	05:00 PM:	5.7 mmol/L	5:00pm:	15 mL/hr	0.2 units
106	05:00 PM:	6.2 mmol/L	5:00pm:	20 mL/hr	0.2 units
107	05:00 PM:	6.6 mmol/L	5:00pm:	30 mL/hr	0.2 units
108	05:00 PM:	6.7 mmol/L	5:00pm:	40 mL/hr	0.2 units
109	05:00 PM:	6.9 mmol/L	5:00pm:	50 mL/hr	0.2 units
110	05:00 PM:	6.9 mmol/L	5:00pm:	60 mL/hr	0.2 units
111	05:00 PM:	6.9 mmol/L	5:00pm:	70 mL/hr	0.2 units
112	05:00 PM:	6.9 mmol/L	5:00pm:	80 mL/hr	0.2 units
113	05:00 PM:	7 mmol/L	5:00pm:	90 mL/hr	0.2 units
114	05:00 PM:	7 mmol/L	5:00pm:	100 mL/hr	0.2 units
115	05:01 PM:	7 mmol/L	5:01pm:	110 mL/hr	0.4 units
116	05:01 PM:	7 mmol/L	5:01pm:	120 mL/hr	0.4 units
		...			
358	05:21 PM:	4.2 mmol/L	5:21pm:	0 mL/hr	15.8 units
359	05:21 PM:	4.2 mmol/L	5:21pm:	0 mL/hr	15.8 units
360	05:22 PM:	4.2 mmol/L	5:22pm:	0 mL/hr	15.8 units
361	05:22 PM:	4.2 mmol/L	5:22pm:	0 mL/hr	15.8 units

Wed Sep 16 05:22 PM: End 16.6 total units Average = 22.1 units/hr

Chapter 3

GLUCOSE SENSOR TESTING

Introduction

As mentioned previously, the Pt/PPD/GOx/Nafion sensor has been extensivley tested on both dogs and humans and showed favorable results. Clinical tests on human volunteers, showed the expected response after an OGTT and that glucose concentrations obained by the sensor closely matched the measured blood glucose levels, reaffirming that the sensors could be used successfully in short-term subcutaneous glucose monitoring [18]. Open-loop tests were performed to observe and record individual sensor characteristics for twenty sensors. The linear responses could be examined for each sensor's defining sensitivity constant and background current (translated to a voltage). Testing then proceeded with the entire setup configured as a closed-loop system with the sensor providing the required feedback signal to the controller. Thus, every component of the model system depicted in Figure 2-2 is incorporated into the final phase of testing.

Open Loop Testing

The sensors were tested *in vitro* to observe their individual response characteristics and to make a comparative study of the sensors as a whole. This phase would establish and confirm the linear operation of the sensor and help judge the suitability of the software controlling algorithm. This

would also allow individual sensor characteristics to be comparitively analysed to determine how closely they correlated. This analysis defines what kind of software calibration, if any, is required before the sensor can be utilized for gathering data that accurately reflects blood glucose concentration.

Five separate glucose solutions of known concentration were prepared in phosphate buffered saline (PBS) and placed in a water bath maintained at 37 °C. The sensors were then connected to the potentiostat-meters and suspended, in turn, in each of the five solutions. Initially, all of the sensors were allowed to stabilize in the solution with the lowest concentration of glucose for 30 minutes (the maximum required stabilzation period previously observed [9]). Readings were then taken at 15-minute intervals as the sensors were moved from one glucose solution to the next in increasing order of concentration. This cycle was repeated up to 20 times for each sensor. The sensors were rinsed in D.D. water at the end of each cycle to remove any residual glucose from the sensor's tip before being reimmersed in the solution of lowest glucose concentration.

Results

Figure 3-1 represents a typical glucose sensor's performance when submersed within five known glucose concentrations of 3, 5, 10, 15, and 20 mM/L of glucose in thermostated PBS at a pH of 7.4. These test conditions were chosen to simulate the typical range of glucose settings a sensor would be exposed to when subcutaneously implanted in a patient. Repeated cycles of testing proved that consistent results could be achieved with most (12) sensors. This is important because the sensor's response characteristic must

be predictable if it is to be viable for prolonged use, as per the requirement outlined in Table 2-1.

Five sensors, however, did not yeild consistent results as demonstrated in Figure 3-2. This is most likely due to the imprecisions accociated with some of the steps in the fabrication procedure. Variations in Pt electrode area and the thicknesses in the tri-layer coating could account for significant deviations in the sensor's response [16]. Damage to the sensor's coating from prolonged exposure to air or sharp surfaces could also account for unpredictable results.

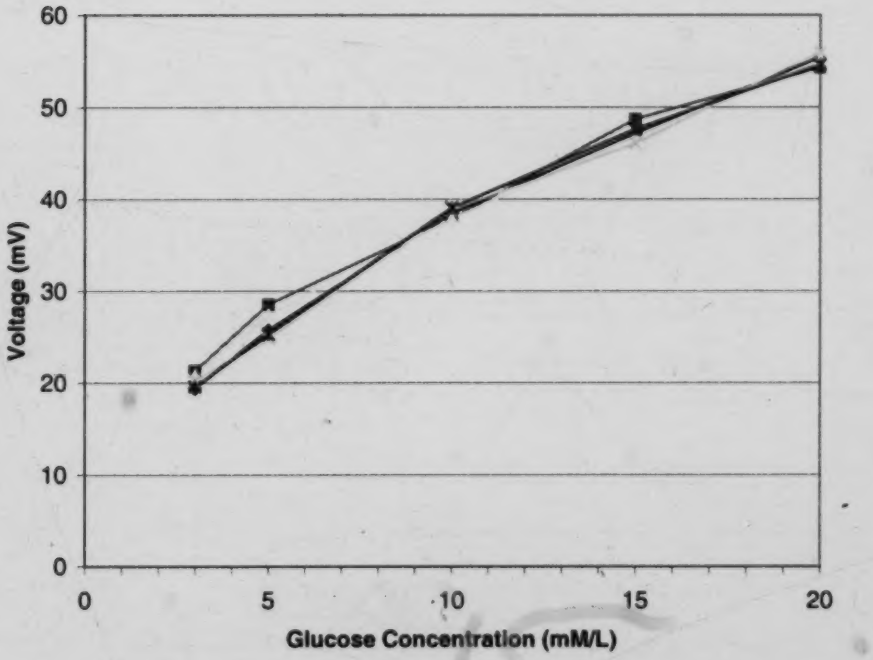


Figure 3-1. Glucose meter voltage readings for known concentrations of glucose in 37 °C thermostated PBS, pH 7.4. A consistent correlation in sensor response was achievable from one measurement cycle to the next. $n = 5$

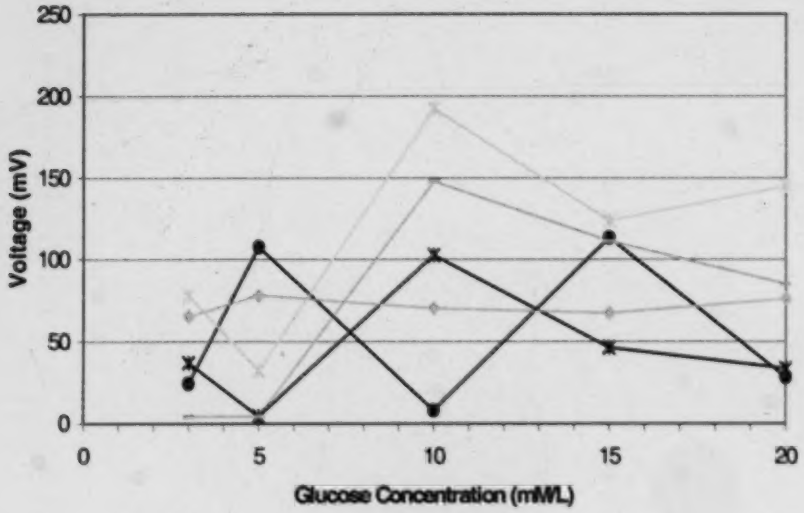


Figure 3-2. Glucose meter voltage readings for known concentrations of glucose in 37 °C thermostated PBS, pH 7.4. Inconsistent and fluctuating response due to imprecise control over fabrication and layer thickness. $n = 5$

The dozen sensor's whose response characteristics demonstrated linear response characteristics were tested 20 times in the five known concentrations of glucose in thermostated PBS and an average behaviour was established for each. The result of this test is shown in Figure 3-3.

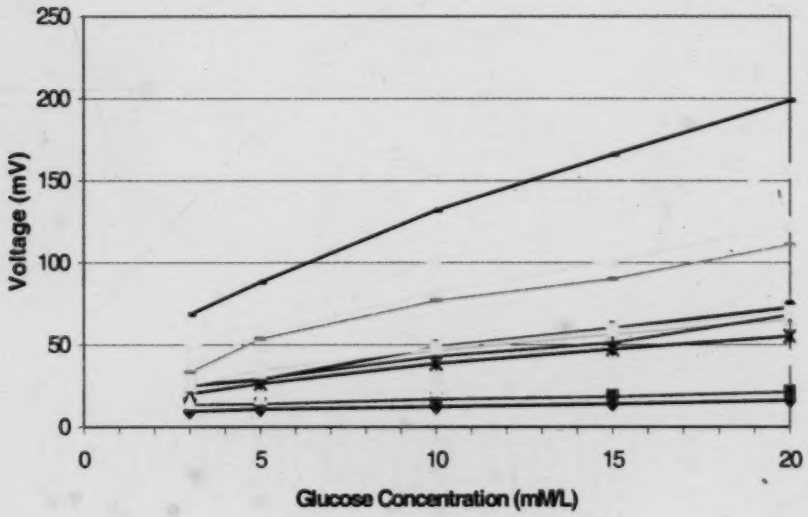


Figure 3-3. Average responses from 12 sensors. Each sensor displayed different linear characteristics.

Although a consistent fabrication procedure was followed for each sensor, unique characteristics resulted for several sensors. Five sensors did however have very similar characteristics. Figure 3-3 also reflects each sensor's sensitivity. A sensor whose characteristic displays a steeper slope is more sensitive to changes in blood glucose concentration. It is this range of sensitivities that indicates the need for a preliminary calibration procedure for

individual sensors before any clinical application can take place. A calibration procedure will inform the controller of the sensitivity of the sensor so that accurate measurements can be made.

An additional test was preformed to observe how a sensor's response would vary after a period of dormantcy; that is, after not being used. The sensor's average characteristic was established using the five known glucose concentrations of 3, 5, 10, 15, and 20 mM/L of glucose in thermostated PBS at a pH of 7.4. The same sensor was then placed in a steril bag and tested again two weeks later. The results shown in Figure 3-4 clearly indicate that the sensor's characteristic had significantly changed, however it's slope, and thus its sensitivity, had depreciated by a lesser degree. A greater difference exists in the voltage measurements made at higher glucose concentrations than at lower concentrations. An even wider gap is predicted if a longer aging period were allowed. Over time the sensor's response becomes almost flat. The paramount implication suggests a limited lifespan that is significantly below the desired 1 year lifespan suggested in Table 2-1.

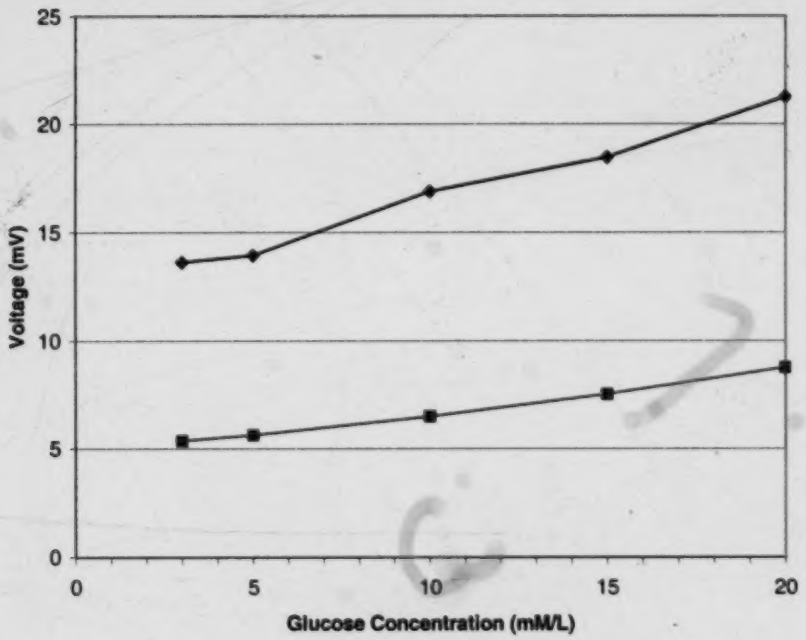


Figure 3-4. Average responses for a sensor with readings taken two weeks apart. (The lower curve shows the latter response.)

After tesing the sensors' characteristics, they where then tested in conjunction with the software controller. Once again, using five known glucose concentrations of 3, 5, 10, 15, and 20 mM/L of glucose in thermostated PBS at a pH of 7.4, stable sensors produced equivalent responses to that shown in Figure 3-5. Prior to running a test, the sensor was calibrated based on its established linear characteristic. For the remaining tests, the potentiostat/glucose meter was connected to a desktop computer via an A/D port on the LabMaster 20009. The LabMaster's parallel interface

with the computer provided the controlling program with the necessary signal from which to calculate and display an actual glucose concentration value. The open-loop experiment provided a means to test the sensor's interaction with the signal amplifier of the potentiostate/meter and the LabMaster interface with the computer. Open-loop testing proved to be successful with the results clearly showing the relationship between the sensor's characteristics and the final observed glucose concentrations now readable from the user interface displayed by the program. With the sensor submersed sequentially in each glucose solution the open-loop results of Figure 3-5 were produced.

Each plateau in the graph of Figure 3-5 depicts the sensor's submersion in a different glucose solution. Samples were taken at a rate of one sample per second, since the sensor's response time is much faster than it is in living tissue, changes could be observed more quickly. The side-effects of such a fast sampling rate however are the spikes produced during the sensor's transition from one solution to the next; i.e., when it was physically removed from one solution and placed into the next one, and which can thus be ignored.

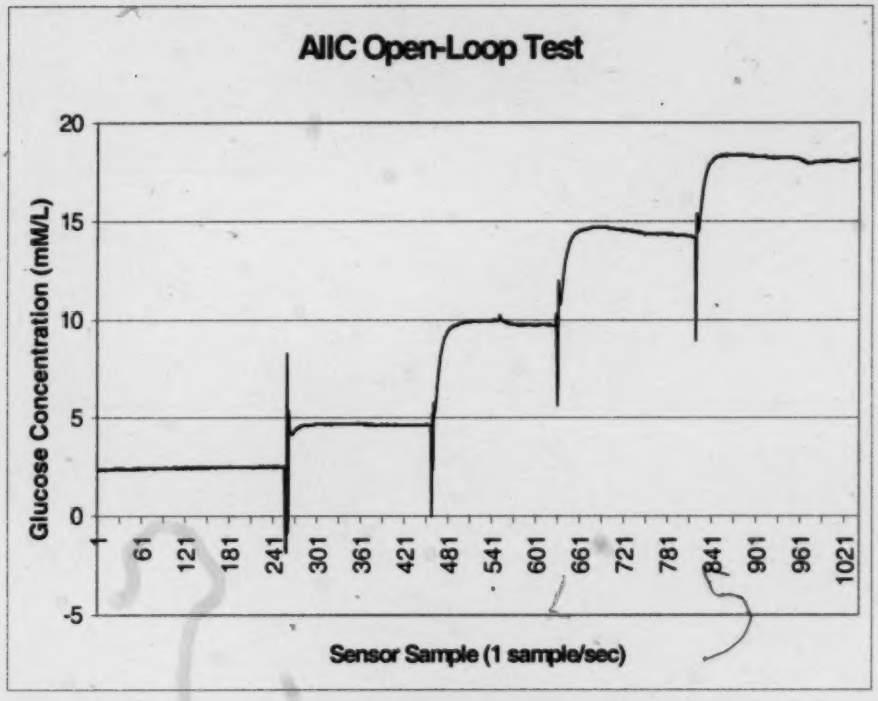


Figure 3-5. Typical *in vitro* response characteristic for a sensor in five known glucose concentrations.

It was observed that the sensor's measurment became less accurate for the highest concentration of 20 mM/L, but overall demonstrated good performance. This occurs because the operating characteristic of the sensor becomes non-linear in extreme glucose concentrations, which in turn is most likely due to reactionary saturation at the sensor's surface.

Closed-Loop Testing

The final step in testing the overall control system must involve closing the loop. This means that the controller must now be permitted to compenstate for elevated levels of glucose by infusing insulin into the patient. Closed-loop testing now involves all of the components indicated in Figure 2-2.

A large beaker with 100 to 200 mL of PBS at a pH of 7.4 with a preliminary glucose concentration of 5.5 mM/L was heated to body temperature and stired at minimum speed using an automatic stirer. A calibrated sensor was connected to the potentiostat/glucose meter which in turn was connected to the A/D port of the LabMaster. The sensor tip was then submersed in the glucose solution. The CIM unit of the variable pressure pump was connected to the serial port of the computer and the pump was equipped with a saline bag with which to infuse the beaker containing the glucose solution. The compensation of elevated glucose would be handled by a method of dilution to simulate the biological metabolization of glucose by insulin. With the system now closed, tests were performed to observe how the human body would react to a relatively sudden increase in glucose that one might experience after a meal. In Figure 3-6 the algorithm's initial infusion rate of 20 ml/hr immediately causes a decline in glucose concentration.

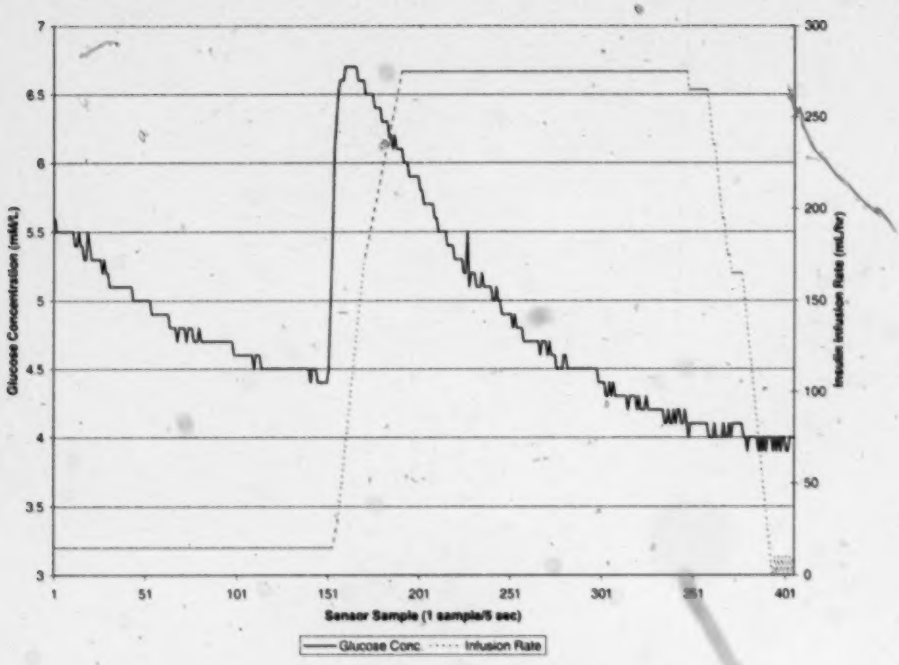


Figure 3-6. Typical closed-loop system response to a rapid increase in glucose concentration introduced to the system after 150 seconds.

The sharp rise in Figure 3-6 indicates the point at which a small amount, 3 to 5 mL, of another glucose soution was injected into the beaker. The sensor immediately registers the change in glucose concentration and the system in turn increases the insulin (in this case saline) infusion rate in order to bring the system back into an equilibrium where the concentration of glucose is brought back into the nominal range of about 4 to 6 mmol/L of glucose. Because the glucose level is reduced gently, it is reduced to the lower concentration of 4.0 mmol/L before the infusion of saline is completely halted according to the given algorithm (Table 2-4).

When glucose concentration was once again reduced to 6.0 mmol/L the speed of the pump was no longer increased as per the given algorithm. This is indicated by the flat region in the infusion rate curve. Because of the one second sampling rate of one sample per second the pump was allowed to reach a pumping rate of 275 mL/hr where it leveled off. A slower sampling rate, more in line with the body's response time of about three minutes, would not have permitted this since there would have been more time between samples for the glucose concentration to change. Thus the pump's speed would be adjusted significantly less often.

The results of another experiment in which two incursions of glucose were allowed to enter the system is shown in Figure 3-7. The initial glucose concentration was preset at 5 mmol/L. Once again, it was gradually lowered by the initial infusion of saline. At the 100 second mark and again at the 315 second mark, a small amount of concentrated glucose solution was added to the beaker. In both cases the infusion of saline diluted the concentration of glucose to within the nominal range before halting. With the first incursion of concentrated glucose, the glucose was lowered faster and thus the pump was halted sooner; when the glucose level was restored to 5 mM/L. After the second introduction of glucose, a more gradual decline in the glucose concentration again caused the pump to cease when a level of 4 mM/L was reached as per the given algorithm (Table 2-4).

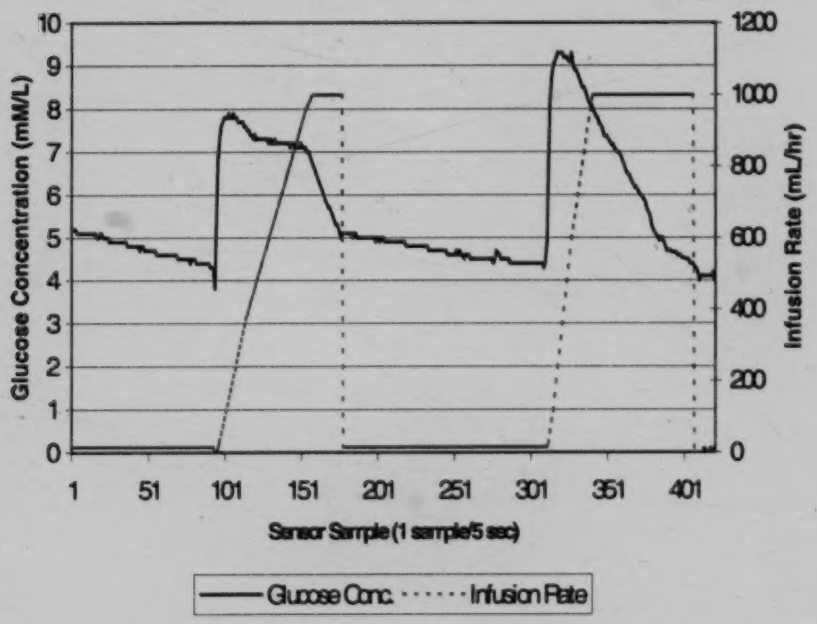


Figure 3-7. Typical closed-loop system response when two rapid increases in glucose concentration occur in succession.

The variable pressure pump reached its maximum pumping rate in this case because a glucose level of 6.0 mM/L was not obtained prior to the pump reaching its top speed of 1000 mL/hr, so incremental increases continued to this point for both incursions of concentrated glucose. Again, a slower sampling rate would mean far fewer incremental increases in the infusion rate.

Chapter 4

CONCLUSIONS

Insulin Infusion Control

The overall closed-loop infusion delivery system was shown to be possible and all components functioned within the prescribed specifications. Stable sensors were shown to reflect and follow with great accuracy, actual glucose concentrations. The closed-loop experiments demonstrated that elevated levels of glucose could be properly compenstated for by a controller that resembles the human body's natural response (Figure 2-3). Using the needle-type, Nafion coated glucose sensor as the feedback element to close the loop the feasibility of such a system was demonstrated.

Canine Testing

Preliminary work was done to conduct open-loop *in vivo* tests on dogs, but proved unsuccessful. On three spearate occations a sensor was implanted into the sub-cutaneous tissue of the neck area of a diabetic dog. In each instance no significant variation could be measured in the sensor's signal current as the when the dog was given an oral bolus of glucose. It is believed that the sensors were either improperly placed or suffered damage upon insertion into the dog's tissue. Even a small nick or scratch could effectivly render a sensor useless. As mentioned ealier, successful trials have been conducted on rats, dogs, and human beings in separate clinical studies.

Future Research

Methods for the treatment of diabetes continue to be diverse. The directions that research has taken ranges from less invasive treatment techniques, such as the insulin inhaler [34, 35] and the jet injector [36] to surgical techniques such as islet transplantation [37, 38, 39]. More exotic methods are also gaining momentum, such as the promising research towards the development of a diabetes vaccine underway at the University of Calgary [40], and tissue engineering for the restoration of damaged or destroyed organ tissue [41]. The methods that have had the most dramatic effect are those that have offered a closed-loop for optimal therapy. One such approach is that of the device-type artificial pancreatic control system discussed in this study.

In 1986, at the Applied Physics Laboratory at Johns Hopkins Hospital University in Laurel, Md., the first programmable insulin-delivery system was implanted in a patient. Some 20 patients since then have received the pumps, which are embedded below the skin in the abdomen. The implanted pump can hold about 2 ½ teaspoons of concentrated insulin [42]. But while the need for daily insulin injections is eliminated for a period of up to three months, the patient must still manually check their glucose levels by conventional means and then transmit the required perscription to the device before it will deliver the insulin. Research has shown that optimal therapy means that the system ideally should respond around the clock as needed to independantly manage blood sugars in real time.

With the current implementation of the automatic insulin infusion control system, further development for its use in clinical environments where central monitoring is employed could be useful. Because the system is computer

controlled, a network of systems connected to a central monitoring station would allow more than one patient to be monitored and treated at a time. The system also provides a useful testbed for alternative control algorithms, glucose sensors, and delivery systems.

However, for portable use, the necessity for the miniaturization of the glucose monitoring component of the system is apparent. MiniMed Inc. (Sylmar, CA) is one company which has been a leader in the development of devices for the treatment of diabetes. Recently (June, 1999), they have received preliminary approval from the Food and Drug Administration (FDA) to commercially distribute a new glucose monitoring system device [43]. Poitout *et al.* have mentioned the architecture of the control unit could be designed to allow for miniturization to the size of a wrist watch [44].

Finally, improvements in sensor life span are crucial for long-term clinical use [10]. This would also help promote the development of a miniature, wearable artificial pancreas. This study should further support the feasibility for a closed-loop control scheme.

Appendix A

ELECTRONIC DESIGN

In order to receive the sensor's signal into the computer a potentiostat was needed to provide a stable bias voltage to the sensor and thus provide the means for a signal current to be produced as per Equation (2-2). For easier open-loop testing the signal current is converted to a voltage and displayed on a small voltage meter. This same voltage is also amplified to provide a more sizable signal to the LabMaster Analog-to-Digital converter for subsequent reading by the computer's controlling software. This appendix outlines the design theory and design steps to produce a portable device that could accomplish all three of these tasks.

A Pontentiostat

A simple potentiostat circuit is illustrated in Figure A-1 below.

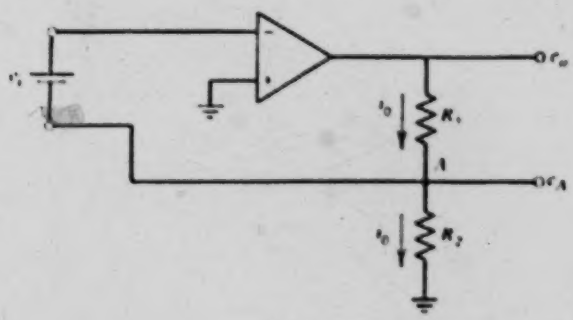


Figure A-1. A circuit for controlling the potential at point *A* independently of the changes in the resistances R1 and R2. [12]

The input voltage applied to the top terminal (the inverting input) of the operational amplifier in Figure A-1 indicates a value of e_i with respect to the bottom terminal (the negative end of the battery). Since the op-amp's inverting input acts as a virtual ground, the lower terminal, and thus point A, are at a potential of $-e_i$ with respect to ground. In order to maintain this condition, the op-amp must adjust its output by contolling currents through the resister network. As a result, this method provides an effective means for controlling the voltage at a certain level, even when the resistances fluctuate during the experiment, and forms the basis for a potentiostat [45].

The current that passes through R_1 also passes through R_2, and thus the total output e_o is $i_o(R_1 + R_2)$. Since, by Ohm's Law, $i_o = -e_i/R_2$,

$$e_o = -e_i\left(\frac{R_1 + R_2}{R_2}\right) \qquad \text{(A-1)}$$

An electrochemical cell, in this case the glucose sensor, can be modelled as a network of impedances like the one shown in Figure A-2 below.

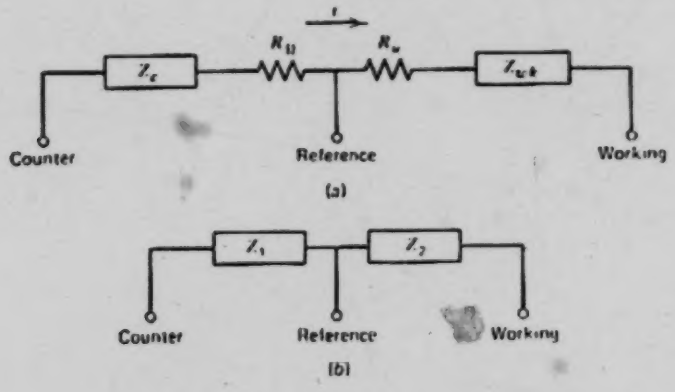


Figure A-2. An electrochemical cell model showing the electrode terminals.

Z_c and Z_{wk} in Figure A-2(a) represent the interfacial impedances at the counter and the working electrodes, respectively, while R_Ω and R_x represent the solution resistances split into two fractions, depending on the position of the reference electrode's probe in the current path. This model is further simplified by the network in Figure A-2(b). While many electrochemical cells require a counter connection, the two terminal configuration of the glucose sensor employed in this control system does not and the network can be further distilled down to a single impedance.

The glucose sensor incoporates working and reference electrodes and it is the impedance between these two that fluctuates during glucose level determination. The circuit, then, that demonstrates the potentiostat for this system is represented in Figure A-3, where interfacial and solution resistances related to the sensor can be represented by the single impedance Z_S, and the output voltage is just the negative of the input supply, or $e_o \equiv -e_i$.

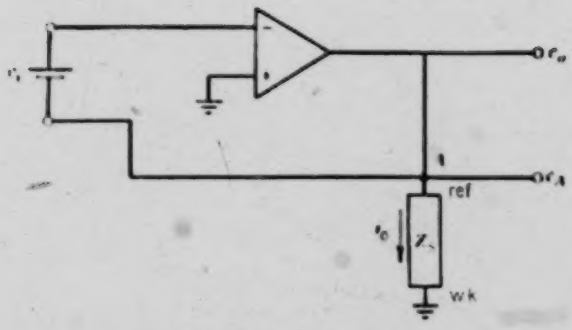


Figure A-3. A simple potentiostat circuit for controlling the potential at point *A* independently of changes in the sensor's impedance Z_S.

The required sensor bias voltage now depends on e_b, but must also be derived from the circuit's 9 volt power supply. The source, represented merely as a battery in Figure A-3, must supply a reference voltage of 0.7 volts. In order to accomplish this a voltage regulator is in order. By definition, the most important requirment of any refernce voltage source is that it must maintain an accurate and stable output voltage with varying conditions of input vóltage, time, temperature, and loading [46].

While the operational amplifier in Figure A-3 is key to the model for a potentiostat, it also bears responsibility for the dual role of a voltage reference source because of its high input impedance and easily adjustable gain. It also incorporates low output impedance and substantial output current capability [47]. In particular, the LM124 low power quad operational amplifier was chosen for these reasons and because it is specifically designed to operate

from a single power supply [48]. Figure A-4 shows the final voltage regulator/potentiostat to provide the necessary sensor bias voltage.

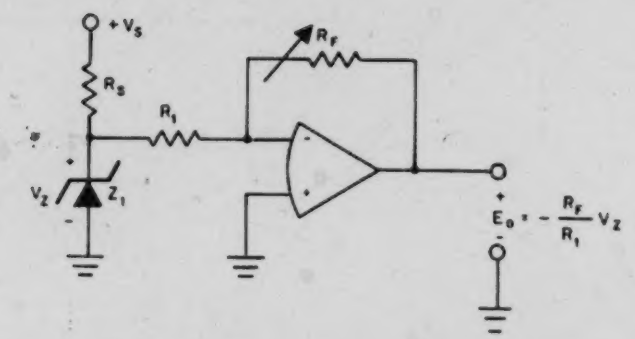


Figure A-4. A potentiostat with a zener referenced source and feedback, which can be adjusted to produce the desired output voltage.

The loading condition on the zener diode is constant and provides excellent regulation even while the input voltage, V_s (a battery) begins to depreciate over time. The output voltage is given by

$$E_o = -\frac{R_F}{R_1} V_{REF} \qquad \text{(A-2)}$$

The required potentiostat voltage output, E_o again is 0.7 volts, to properly bias the glucose sensor. Measured from the output terminal to ground in Figure A-4, the output will be -0.7 volts since the circuit is configured in an inverting mode of operation. A 3.3-Volt zener diode will act, in its common role, as the voltage reference. Thus from Equation (A-2),

$$-0.7\,\text{V} = -\frac{R_F}{R_1} 3.3\,\text{V}$$

and

$$\frac{R_F}{R_1} = 0.212$$

A combination of resistances is needed to meet this ratio and the following values were chosen: R_F = 500 Ω variable and R_1 = 1.8 kΩ. Resistors with 5% tolerances were satisfactory since the adjustable feedback resistor, R_F (a trimmer potentiometer), would be used to calibrate the potentiometer output to exactly 0.7 volts. The trimmer potentiometer also offers the advantage of being able to adjust the bias voltage to different values to accomodate the requirements of different kinds of sensors. With the particular resistor combination chosen this would allow for an effective range of 0 to 0.917 volts for a possible sensor bias voltage.

The reference voltage component is, for all practical purposes, isolated from any load being driven. The effective output impedance can thus be derived as

$$R_{OUT} = \frac{R_o}{A\beta} \qquad \text{(A-3)}$$

where

$$\beta = \frac{R_F}{R_1}$$

R_o is the open-loop output impedance of the operational amplifier and A is its open-loop voltage gain. The load regulation can thus be written as

$$\text{Regulation } \% = \frac{R_o}{A\beta R_L} \times 100 \qquad \text{(A-4)}$$

Ideally, R_o for any op-amp is assumed to be very small (zero for all intensive purposes), but a typical value of 75 ohms is assumed for design purposes [49]. The open-loop voltage gain is 100 dB, or 100 000 [15].

While the supplied bias voltage must remain constant, the sensor's impedance is allowed to fluctuate inverse-proportionally with the concentration of hydrogen peroxide in the host's blood. Put another way, this fluctuating impedance is reflected by the fluctuating current produced by the reaction of Equation (A-2). It is this current that in turn reflects the concentration of glucose in the host's blood. The glucose meter is equipped with an LCD display to indicate a value that is proportional to the glucose concentration in which the sensor is placed. The meter component itself is a voltmeter capable of displaying a voltage in the range 0 to 200 mV. Thus, the current available from the sensor must be converted to a voltage. This is done with a resistor placed across the sensor's working electrode to ground. The voltage potential across this resistor is then measured and displayed. The maximum current the sensor will practically produce is approximately 500nA (corresponding to a

high concentration of blood sugar levels). Thus in order not to exceed the 200 mV range of the meter, a resistance of about 400 kΩ is in order, by ohms law. By design then, and from Equation (A-4), the regulation is limited to a value smaller than 0.001 %, providing for a very stable potentiostat.

Voltage Amplification

In order to accomodate the computer interface, a degree of signal amplification is needed to satisfy the sensitivity requirements of the LabMaster analog-to-digital convertor. An output swing of 0 to 3 volts was chosen as the desired range so as not to exceed power source limitations even after the source has decayed. With the potential across the current-to-voltage transforming resistance providing the input, a conventional non-inverting amplifier with a gain of 15.3 was employed. Thus the 200 mV peak across the afore mentioned resistor would be amplified by the gain value to produce a peak output value of just over three volts, meeting the LabMaster's requirement.

Figure A-5 shows the schematic diagram of the completed potentiostat/meter/amplifier. Though the possibility of electrical surges is unlikely, an extra stage, such as an opto-coupler, could easily be included to provide electrical isolation between power sources associated with the computer and the implanted sensor as a precaution before initiating human clinical studies.

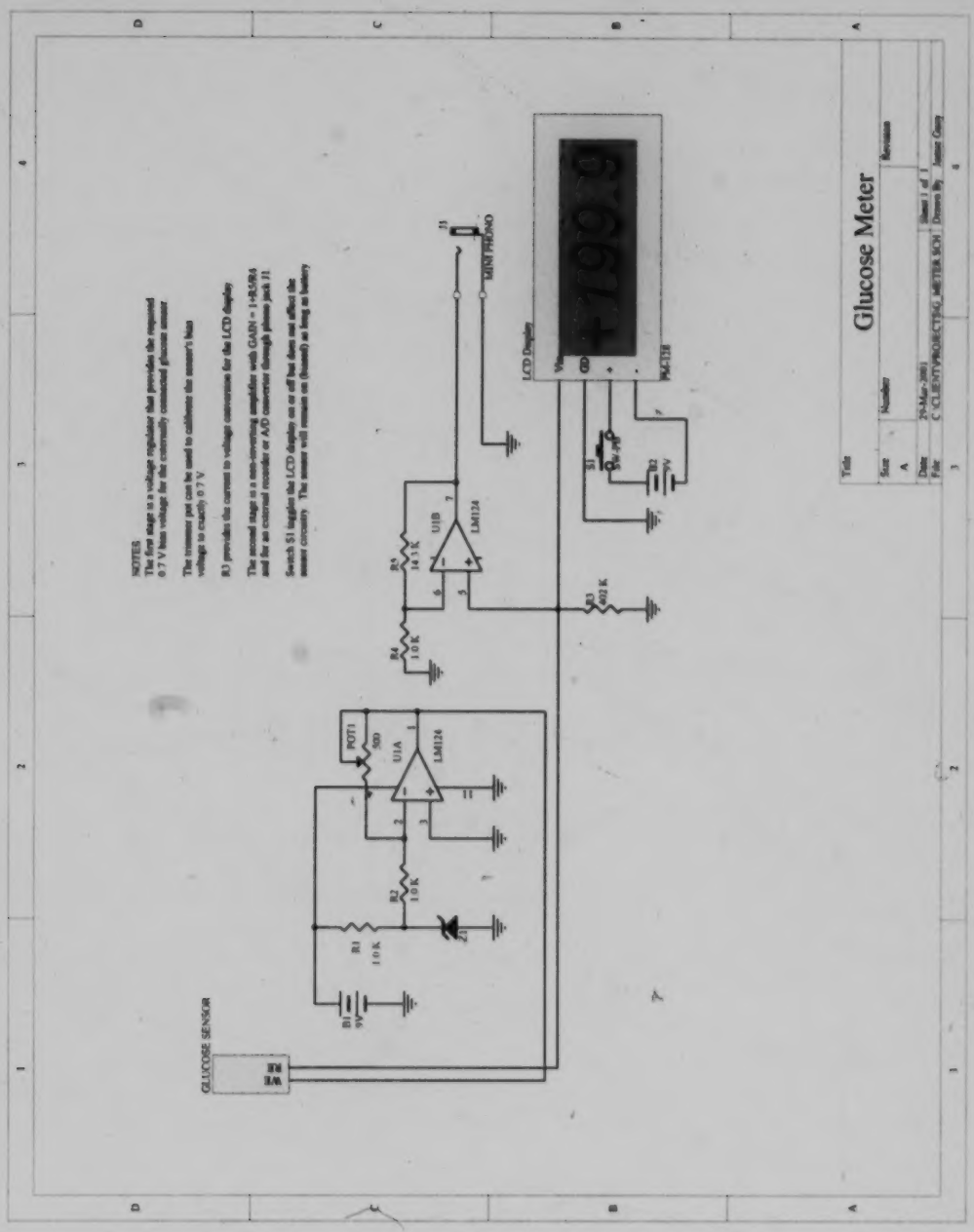


Figure A-5. Schematic diagram of the completed potentiostat/meter/amplifier.

Appendix B

SOFTWARE DESIGN

The core of the closed-loop insulin infusion system is the controlling software, or program. A program is an organized list of instructions that, when executed, causes the computer to behave in a predetermined manner, and in a way is like a recipe. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables. The variables can represent numeric data, text, or graphical images.

While there are many languages in which a program may be written, C was chosen because of its powerful and flexible control capabilities over a computer's hardware components including the serial and parallel ports. It was also chosen because of its relatively low developmental overhead and application towards modular design. The advantage is a shorter development period and a program with a quicker execution speed. A modular design approach also allows for different program components, or modules, to be replaced to accommodate different hardware components should those components be replaced. For example, if a different infusion pump model were to be used, a different code module could be developed without disruption to the rest of the program code.

Design Approach

The software development process involved the following steps: problem identification and analysis, user consultation, planning, coding, debugging, testing, and implementation. As is usual in software development, several, if not all of these steps, are repeated as necessary as a project evolves.

The original problem presented was that of providing a means by which a computer program could sample glucose concentration levels via a needle-type glucose sensor, determine an appropriate insulin infusion rate based on that reading for one of three medical situations: islet transplant, a surgical procedure, or delivery, and then instruct a pump to carry out the actual infusion. In addition, all data would be recorded for future analysis and displayed in real-time to the user.

In order to solve this problem it was broken down into three separted stages to simply the overall development. Each stage produced a different operational mode for the control system. The first stage produced a program for open-loop operation in which glucose levels would be gathered manually on a regular schedule and then entered into the computer using the keyboard. The collected data (both glucose levels and corresponding insulin infusion rates) was then displayed in a meaningful way on the computer's monitor for medical staff, who would then program a variable pressure pump to deliver the insulin at an appropriate rate. This first implementation of the program incorporated the required glucose level-to-infusion rate algorithm and established the primary interface elements that would carry through to other implementations. The second stage extended the capabilities of the first by establishing a serial communications link with an IVAC 570 variable pressure

pump equipped with a Computer Interface Module (CIM). Thus, the program user would enter glucose levels manually, but insulin infusion would be computer controlled. The third stage added the final component that would form a closed-loop system – the glucose sensor. The current signal from the sensor was conditioned using the potentiostat/amplifier/meter described in Appendix A, then sampled and digtized using the analog-to-digital converter of the LabMaster 20009, which in turn was received by the computer program via a parallel port.

At the end of each stage in the software's development the steps previously mentioned were conducted until the application was ready for practical use, beginning with problem identification and analysis and ending with implementation.

Code Structure

A modular, procedure-oriented, top-down programming approach was taken to develop the actual code. The program was compiled using the Turbo C++ integrated development environment (version 3.0) made by Borland® and will execute on any personal computer running the MS DOS™ or Windows™ operating systems.

The modules contain the various functions for controlling different aspects of the program's operation as well as the hardware elements that are infaced with the computer. The table below summarizes the contents of each program module.

Table B-1 Summary of the functions contained within the program modules for the controller software.

Program Module	Description
Ctime	Time related functions for entering and displaying times and dates and keeping track of countdowns
Gio	Functions for collecting data in a graphic display mode as well as displaying prompts, menu options, error messages, and providing cursor control
Graphics	Functions for initializing the graphics display mode and displaying the special prompts, and warning messages
Infusion	Contains the "main" function and the process control loop that detects and executes all actions
Ivac	Functions for constructing valid instructions to send to the IVAC 570 variable pressure pump
Labmastr	Functions for initializing the LabMaster 20009 A/D converter, as well as functions for sending and receiving information to and from it respectively
Menu	Functions for handling user input fields and menus in text mode
Misc	Additional functions for displaying input prompts and messages and receiving user input in a graphics mode and while in a loop where other processes may be going on at the same time.
Process	Functions which determine the appropriate insulin infusion rate for the three medical situations: islet transplant, surgery, or delivery
Serial	Functions for sending and receiving characters through a serial port
Sound	Functions for handling the auditory warnings

Program Module	Description
Store	Functions for initializing, opening, and writing to a text file to store patient information as well as times, measured glucose levels and insulin infusion rates
Terminal	A custom terminal program for sending instructions directly to the IVAC 570 pump
Title	Displays an initial graphic title screen when the program is run
User_in	Functions for collecting the patient's personal information including name, weight, hospital, physician, as well as a function for setting the name of patient's actual data file where the information is stored

Serial Interfacing with the IVAC 570

The IVAC 570 infusion pump can be equipped with a Computer Interface Module (CIM), which essentially acts as a modem by establishing the serial communications link between the pump and a personal computer via a standard serial cable. This allows software running on a separate device (a computer in this case) to inialize the pump and set its infusion rate remotely.

When the CIM receives a properly formatted command, it returns an acknowledgement to the computer. There are two types of command for controlling the CIM: action and non-action commands. Once a communications link is established with the CIM, it must receive a Mode Interrogation command (non-action command) within a specific time-out period, which can be configured to range from 20 to 120 seconds. If the CIM does not receive this command withing the tim-out period, the pump will invoke a an alarm message (both audible and textual using its own display)

and the communications link will be broken. Non-action commands do not change the operating status of the pump, but merely report on it.

The command sent to the CIM must have the following format, otherwise its acknowledgement response will indicate an error.

Xdddxxabcd[cr][lf]

The command must begin with an upper case "X". The string of characters, "ddd", denotes the ID number of the CIM, which must be a three-digit number between 001 and 999. This is to account for the possibility that more than one CIM could exist within a networked environment. In such a case commands are sent to CIM units individually. The command code is denoted by "xx". The command code may be either one or two characters in length, and specifies the precise action to be taken by the pump or the data to be returned if it is a non-action (interrogation) command. The string "abcd" represents the operand field that appears in many (but not all) commands. This field is used to specify a particular command-associated value, such as the flow rate. Non-action commands rarely have operand fields, and are usually followed by a question mark (?) to indicate that the command is a request for information only. As indicated in the general command syntax above, all properly formatted commands are terminated by an ASCII carriage return [cr] and line feed [lf]. The CIM will validate any commands it received on its own. If a properly formatted command has been received, it will return an aknowledgement to the computer with the following format:

Adddyyabcd[cr][lf]

The capital letter "A" and the "ddd" indicate an acknowledgment and the ID of the CIM respectively. The string "yy" is the response code. Some non-action commands display an operand field for the string "abcd". Below is a typical command that would be sent to the CIM to change its infusion rate and the acknowledgement that would be returned.

X001RP0700[cr][lf]

A001OK[cr][lf]

The "RP" command sent to the CIM unit 001 will change the pump rate to 70.0 mL/hr. The CIM acknowledges the properly formatted command with an OK message for the same unit. In order to detect a reliable serial connection between the CIM and computer a Mode Interrogation command is sent every 20 seconds. After it is received an acknowledgement is returned indicating the pump's current operating status.

X001M?[cr][lf]

A001M17R[cr][lf]

In this case, CIM unit 001 reports a code of "17R". The "1" means that the pump is currently in the "RUN" operating mode, the "7" indicates that the pump's display is showing units of "mL/hr", and the "R" means that the pump is being controlled remotely (as opposed to locally, without external control). If the pump were experincing an alarm condition a "5" would have been returned instead of a "1" and the following character would indicate which condition was causing the alarm. If this were the case, an appropriate

message would be displayed to the system user on the computer's screen and an audible alarm sounded so that the problem may be immediately dealt with or corrected. Normal operation could then resume afterwards.

The C-code function used two send instructions to the IVAC and receive characters back from it is show below.

```
/* SOURCE: ivac.c */
/* These functions provide serial interfacing with
/* the IVAC 570 infusion pump. */

#include <stdlib.h>  /* Contains itoa()     */
#include <string.h>  /* Contains str...()   */
#include "ivac.h"  /* Contains CR and LF */
#include "serial.h"

/*
** PROTOTYPE: void ivac570(int unit, char *command)
** PURPOSE: Send a command to an IVAC 570 Variable Preassure Pump
** NOTE: unit must be in the range 1-999
*/
void ivac570(int unit, char *command, int value)
{
 char ivac_command[20]; /* Stores the complete ivac command string */
 char unit_str[5];
 char *null_ptr;   /* Points to the NULL at the end of a string */
 char value_str[5];
 int command_length, i, ch;

 /* Convert unit number to three character string */
 unit += 1000;   /* Add 1000 to pad left side with zeros */
 itoa(unit, unit_str, 10);/* Convert to string */

 /* Build command string */
 strcpy(ivac_command, "X");
 strcat(ivac_command, unit_str+1); /* Add 1 to point to first digit */
 strcat(ivac_command, command);      /*   of IVAC unit number */

 /* Check if command requires a value */
 if (!strcmp("RP", command)) {  /* Set Infusion Rate */
  value = value * 10 + 10000;
  itoa(value, value_str, 10);
  strcat(ivac_command, value_str+1); /* Append value */
 }

 command_length = strlen(ivac_command)+2;/* Get length of command string */
          /* Add 2 for the CR and LF */
```

```
/* Replace the NULL terminator with CR and LF */
null_ptr = strchr(ivac_command, NULL);
*null_ptr = CR;
null_ptr++;
*null_ptr = LF;

/* Send command to IVAC */
for (i=0; i<command_length; i++)
 s_sendchar(*(ivac_command+i));

/* Receive response from IVAC - check if character waiting */
while ((ch = s_rcvchar()) != -1) {
 putch(ch);     /* display char received from DTE */
}
}
```

Parallel Interfacing with the LabMaster 20009

The LabMaster 20009 is a combination Analog-to-Digital/Digital-to-Analog convertor. The daughter board of the devices is plugged directly into an IDE slot of the computer whild a parllel cable connects it to the external connector box. Custom functions were developed to send initialization commands to the LabMaster as well as read digitized values of the sensor's conditioned current signal. The code listing below shows how this was accomplished.

```
/* SOURCE: labmastr.c
 * Functions to control I/O for the LabMaster 20009
 */
#include <dos.h>
#include "labmastr.h"

int high, low;


/* PROTOTYPE: void labmaster_init(int gain)
 * PURPOSE: Initialize the Lab Master
 * PARAMETERS: gain: An integer of 1,10,100, or 500 that is used
 * to set the gain of the Lab Master
 * NOTES: Bits 1 and 0 (GS1 and GS0) of the control byte are
 *  used to set the gain. If the Lab Master is not equipped with
 *  software programmable gain, these bits are ignored.
```

```
 */
void labmaster_init(int gain)
{
 unsigned char control_byte;

 control_byte = 0x80;     // Disable auto-incrementing, external start
                          //  conversions, and all interrupts
 switch (gain) {
  case 1:
   control_byte |= 0x00; // gain = 1
   break;
  case 10:
   control_byte |= 0x01; // gain = 10
   break;
  case 100:
   control_byte |= 0x02; // gain = 100
   break;
  case 500:
   control_byte |= 0x03; // gain = 500
   break;
  default:
   control_byte |= 0x00; // set gain to 1 for invalid gain parameter
   break;
 }
 outportb(LABMASTER+4, control_byte);
}


/* PROTOTYPE: void labmaster_out(int dac_port, int data)
 * PURPOSE: Send data to the LABMASTER 20009
 * NOTES: The parameter dac_port may be 0 or 1, the parameter data should
 *  be between -2048 and 2047 (signed 12-bit integer).
 */
void labmaster_out(int dac_port, int data)
{
 high = data>>8;
 low = data & 0x0ff;
 if (dac_port == 0) {
  outportb(LABMASTER+1, high);  // Send the upper 4-bits first
  outportb(LABMASTER, low);    // Send the lower 8-bits
 }
 else {
  outportb(LABMASTER+3, high);  // Send the upper 4-bits first
  outportb(LABMASTER+2, low);  // Send the lower 8-bits
 }
}


/* PROTOTYPE: int labmaster_in(int adc)
 * PURPOSE: Sample from LABMASTER 20009
 * NOTES:  The parameter adc is the ADC input channel to set the
 *  multiplexor to before conversion.  A function call will start a
 *  conversion and then wait for the conversion done signal.
 *  The value returned will be between -2048 and 2047 (signed
```

```
 *  12-bit integer).
 */
int labmaster_in(int adc)
{
 int data;

 outportb(LABMASTER+5, adc); // Initialize adc channel in the analog
multiplexer
 outportb(LABMASTER+6, 0x00); // Start a conversion

 /* Wait until the highest bit in the register becomes 0,
  *   indicating the end of a conversion */
 while (inportb(LABMASTER+4) < 0x80)
  ; // Do nothing

 low  = inportb(LABMASTER+5); // Read the 12-bit (8+4) data
 high = inportb(LABMASTER+6);

 /* Obtain the actual decimal magnitude */
 data = high*256 + low;
 return (data);
}
```

Process Control

The programs operation is broken down into processes. Since the software must function in a real-time fashion, there can be no process which causes the program to halt. The principle challenge of the program's design was in developing procedures that allowed for a *flow-though* approach. Thus, even when a prompt requiring user input on the keyboard is displayed other processes are still receiving attention constituing multiple *threads* in the program's execution. This means more than one thing are happening at a given moment, such as updating the time, or the display, or the data file, all while a user may be interacting with the system. The flowchart of Figure B-1 below summarizes the program's various processes.

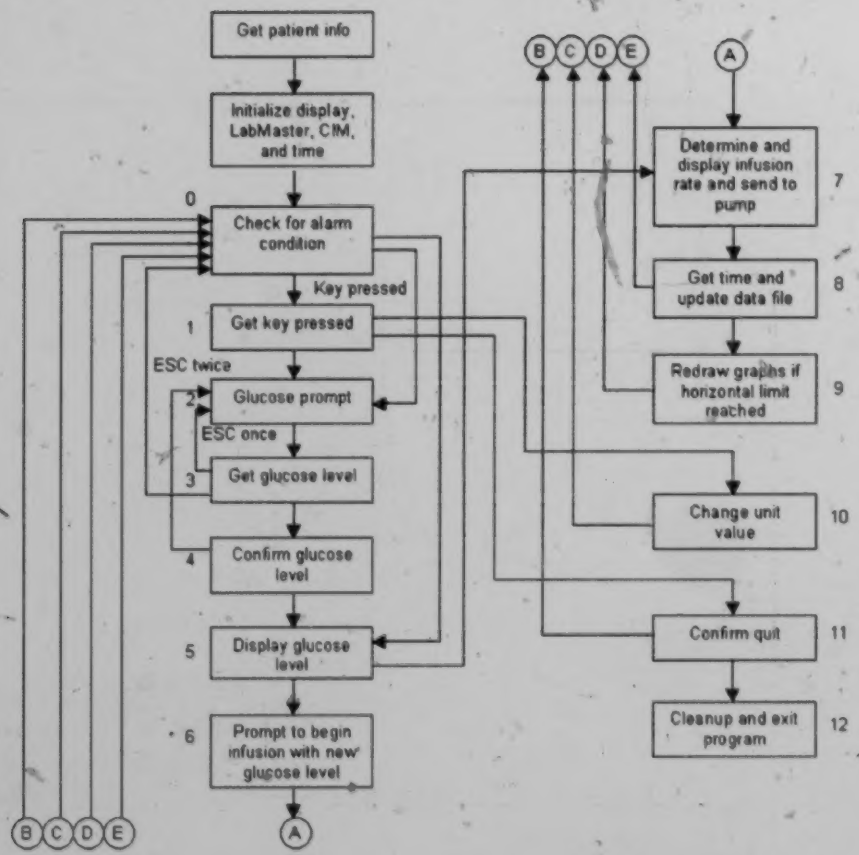


Figure B-1. The process threads that execute in different combinations depending on the state of external hardware or user interaction.

The numbers next to the blocks correspond to the numbers used to direct program flow using the "switch" statement shown at line 388 of the program listing at the end of this appendix. Note that processes 1 to 4 are only executed if the user is entering glucose levels manually using the computer's keyboard as opposed to obtaining a glucose measurment from the sensor.

Error Protection

Since the insulin delivery system is a safety-critical system, appropriate features have been built into the software to sense and report when a problem has occurred, and if necessary (and possible), to take action to prevent injury. Thus, a limited measure of *fault-tolerance* is incorporated into the system. Fault-tolerance is a system's ability to react positively to its own failure. However, due to the integrated nature of the system, overall reliability depends on the reliability of every component, and no one can ever be certain that any system will be 100% fault-free and fault-tolerant. A fault-tolerant software system may not necessarily be safe. The sources for a system malfunction in this case basically stems from three considerations. Firstly, the specification for the system may be incomplete. Extreme or unforeseen circumstances requiring special attention may arise not originally specified. Secondly, hardware malfunctions may cause the system to behave in an unpredictable way, presenting the software with an operating environment that was unforeseen. For example, the software, acting as the controller by determining the necessary insulin infusion rate, will still operate reliably even when the characteristic of the glucose sensor has changed. The controller would continue to operate using the initial report of the sensor's operating characteristic, resulting in incorrect infusion rates. This problem would have to be overcome with a manual re-calibration or replacement of the sensor. Theoretically, with a more sophisticated analysis of the data as it is being received, the sensor's characteristic could be analyzed and compensated for dynamically by the computer. Thirdly, the system operator may offer input to the system that causes a malfunction. Unintentionally, the user may perform an action that the software will obey, but may be inappropriate [50].

In this case, the user could start the infusion process before the IV needle is properly inserted into the patient.

A number of steps were taken during the development process to prevent some of the afore mentioned problems from occuring. The first source for malfunction arises out of a lack of information in developing the specifications for how the system should operate. Because the system must follow a strict medical protocol for handling high (or low) levels of blood glucose levels, medical personnel were consulted on several occasions during the development stage. The same instructions for determing the necessary insulin delivery rates via an infusion pump were provided by trained staff at the University of Alberta Hospital. It was this protocol upon which the software algorithm was based in setting infusion rates. They were also consulted on how patient information should be gathered and displayed by the computer. The design of the software's graphic user interface must be simple enough to use without overwelming the user, while complex enough to provide sufficient operability for provide practical care of a diabetic patient. This would include everything from the software's operation to its abilitity to collect and record accurate information for later analysis. Any such system, of course, must undergo scrutinous *in vitro* testing to make sure that the specifications are indeed met to the satisfaction of medical staff, and before any possibility of human trials could be considered.

The second problem of possible hardware malfunction is also dealt with. Should a power failure occur, devices not connected to an uninterruptable power supply (UPS) will cease to function. The computer will not longer be able to interrogate the CIM as a result of the broken communications link.

The pump, however, is equipped with a battery which will take over in such a situation. The resulting loss of the computer's connection will cause the IVAC pump to display an error message and sound an audible alrarm to alert the user or medical staff. Any error in the computer's hardware, including the internal LabMaster circuit board would result in a signal loss from the sensor and would also generate an alarm condition, since a glucose level is not allowed to be zero. Should the sensor fail such that it produced erratic readings, the computer would generate an alarm and halt the infusion pump, thus preventing it from infusing incorrect amounts of insulin. The software can detect such a condition by comparing the current glucose reading with the previous one. If the reading changes too rapidly, outside of a normal change gradient for glucose, an alarm condition will be generated. Again, a message is generated as well as an audible alarm. The software cannot detect incorrect glucose readings from the sensor if they are inside nominal ranges however.

The third possiblility for error arises from incorrect user input. The possibility for accidental input for a glucose level is minimized, and practically eliminated, since two levels of confirmation must be accepted by the user before it can be entered. Once again, if the user entered a glucose value the was dramatically and unnaturally high an alarm condition would still be generated and the infusion pump halted.

Main Module Program Listing

A program listing for the most important module, "Infusion", is shown as the end of this section. The Infusion module initializes hardware, starts the

process of collecting information, and executes the main loop. The main loop allows the program to branch from one process to another, executing different sections of code depending on the user's interaction with it or on the status of a peripheral device.

Each line of code is preceded by a number and a colon (:) character for reference purposes, but it is not part of the actual source code.

Lines 1-21 are preprocessor directives that are carried out before the source code is actually compiled. These directives perform the tasks of incorporating the contents of other files into the source file and replacing string patterns or numbers with another substitutionary string. Lines 23-74 are global variable and function declarations. "Declarations" state which variables and functions will be used in the program as well as the type of information they will store or use. Two of the most important variables are specified on lines 31 and 32.

```
float glucose_level[ARRAYSIZE];
int insulin[ARRAYSIZE];
```

The first variable represents an array of floating point numbers. This array stores the glucose levels read into the computer via the LabMaster. The second variable is an array of integers, which stores the insulin infusion rate that must be sent to the CIM unit of the IVAC pump. "ARRAYSIZE" represents a number for the size of the arrays, or in other words, the number of values it can store.

Lines 76-104 are local variables for the "main" function, which begins on line 75. Local variables are valid only inside the function, in which they are

declared. The body of the "main" function extends from line 76 to line 997. The remaining lines are for a global function which rounds off floating point numbers to a precribed number of decimal palces. The main process loop begins on line 185.

```
/*****************************************************************
 *                                                               *
 * SOURCE:  INFUSION.C - insulin infusion rate/concentration     *
 *     monitor for islet transplant, surgical operations and     *
 *     delivery.                                                 *
 * VERSION: 1.0                                                  *
 * AUTHOR:  Jamie Guay, (Department of Electrical and            *
 *     Computer Engineering, University of Alberta,              *
 *     Canada)                                                   *
 * DATE:    May 2, 1995                                          *
 * PURPOSE: This program receives the current glucose level      *
 *      measured from a patient and then outputs the proper      *
 *      insulin infusion rate. A data file is created and        *
 *      updated and includes the patient's personal              *
 *      information as well as the current time, glucose,        *
 *      and insulin infusion rate at the beginning of each       *
 *      specified interval of time for the islet transplant      *
 *      procedure.                                               *
 *          Since time is an integral part of the program,       *
 *      the computer's system time should be correct.  The       *
 *      date and time are displayed and confirmed by the         *
 *      user at the beginning.                                   *
 * USAGE:   Enter the program name at the DOS prompt or          *
 *      click on the icon in Windows.                            *
 * COMPILER: Borland Turbo C/C++ Version 3.0                     *
 * REFERENCES: Barkakati, Naba. The Waite Group's Turbo C++      *
 *      Bible, Indianapolis, IA: Sams Publishing, 1994           *
 * ERROR HANDLING: Graphic and text file calls will              *
 *     terminate the program if they fail and provide            *
 *      appropriate error messages.                              *
 *                                                               *
 *****************************************************************/

  1:  #include <bios.h>      /* Contains bioskey()  */
  2:  #include <conio.h>     /* Contains kbhit()  */
  3:  #include <ctype.h>     /* Contains toupper()  */
  4:  #include <dos.h>       /* Contains delay()  */
  5:  #include <graphics.h> /* Contains setcolor()  */
  6:  #include <math.h>      /* Contains fabs()   */
  7:  #include <mem.h>       /* Contains memset()  */
  8:  #include <stdlib.h>    /* Contains exit()   */
  9:  #include <string.h>    /* Contains strcat()  */
 10:  #include <time.h>      /* Contains time(), difftime() */
 11:  #include "infusion.h"
 12:  #include "menu.h"
 13:  #include "myfields.h"
 14:  #include "ivac.h"      /* Contains ivac570() */
 15:  #include "serial.h"
 16:  #include "labmastr.h" /* Functions for the LabMaster 20009 */

 17:  #define PORT 1          /* Comm port 2 */
 18:  #define LM_MAXNUM 2047 /* Maximum digital value of the
LabMaster */
```

```
  19:  #define LM_MAXRNG 10.0 /* Maximum voltage range of the
LabMaster */

  20:  extern unsigned _stklen = 32768U; /* Increase Stack Size to
32k */

  21:  extern FieldList myfields; /* Defines user input fields */

  22:  /***** Declare Global Variables *****/
  23:  TextData *patient_info;
  24:  char *menu1_choices[] = {"Islet Translplant",
  25:         "Surgery",
  26:         "Delivery"};
  27:  char *menu2_choices[] = {"Manual Control",
  28:         "Infusion Control",
  29:         "Sensor + Infusion Control"};
  30:  int control_type;  /* Computer control type */
  31:  float glucose_level[ARRAYSIZE];
  32:  int insulin[ARRAYSIZE];   /* Rate in mL/hr or Concentration
in units/500 mL */
  33:  char first_time_label[15];
  34:  char glucose_time_label[ARRAYSIZE][15];
  35:  char insulin_time_label[ARRAYSIZE][15];
  36:  int glucose_barcount;
  37:  int insulin_barcount;
  38:  int hour_count;    /* Elapsed hours, incremented on the hour */
  39:  int glucose_barwidth[ARRAYSIZE];/* Width of bar graph bars */
  40:  int insulin_barwidth[ARRAYSIZE];
  41:  int yellow_barwidth[ARRAYSIZE];
  42:  int xinput, yinput;   /* Coords for user input */
  43:  int xcoord1, xcoord2;
  44:  int save_xcoord1, save_xcoord2; /* Last coord plotted before
Units Graph displayed */
  45:          /* Pointer to stored image of graphs */
  46:  int xlimit1, xlimit2;  /* Absolute horizontal graphing limits */
  47:  int ylimit;       /* Absolute vertical graphing limit */
  48:  float scale1, scale2, scale3;/* Glucose, insulin, and total
units graphing scales */
  49:  char far *image_buffer1, *image_buffer2;
  50:  unsigned int image_bytes1, image_bytes2;
  51:  int prompt_color = WHITE;
  52:  double float_value;
  53:  int alarm_interval = 60; /* Initial alarm set at 60 minutes
*/
  54:  BOOL ALARM_ON = FALSE;
  55:  BOOL NO_ALARM = FALSE;  /* Set if alarm shut off with down
arrow key */
  56:  BOOL CHECK_ALARM = TRUE;
  57:  BOOL ENTRY_LINE_CLEARED = FALSE;
  58:  BOOL START_TIMEOUT = TRUE;
  59:  BOOL SHOW_UNITS = FALSE; /* True if 24 hr units graph
displayed */
  60:  BOOL TIMEOUT_5 = FALSE;  /* True if delaying audible alarm
for 5 min */
```

```
  61:  BOOL MODE_INT_SENT = FALSE; /* True if a Mode Interrogation
code sent to IVAC
  62:          in the last second */
  63:  BOOL SAMPLE_SENSOR = FALSE;

  64:  time_t current_seconds, timeout_start_seconds,
interval_start_seconds;
  65:  int start_minute = 99;
  66:  int current_minute;
  67:  int current_hour;
  68:  struct tm *local_time;
  69:  float units_per_hour[500]; /* Total hourly units */
  70:  float average_units;  /* Average hourly insulin units */
  71:  struct tm *myclock;

  72:  float sensorK;   /* Constant of proportionality from sensor
calibration */

  73:  // Function Prototypes
  74:  float round(float real, int places);

  75:  void main(void)
  76:  {
  77:   /***** Declare Local Variables *****/
  78:   int i, j;
  79:   int gmode;    /* Graphics mode number */
  80:   char patient_name[26];
  81:   time_t  start_seconds, last_update_seconds;
  82:   int last_minute = 99;
  83:   int last_hour;

  84:   time_t this_second, prev_second; // used to time Mode
Interrogation commands for pump
  85:   int current_second, last_second = 99; // FOR DEMO PURPOSES ONLY

  86:   int current_day, last_day = 99;
  87:   char time_str[50], time_msg[80], clock_str[10];
  88:   double time_difference;
  89:   int infusion_count;      /* Total infusions */
  90:   int keypress;
  91:   int process;
  92:   float insulin_units = 0.0;
  93:   float total_insulin_units = 0.0;
  94:   int total_bar_height;
  95:   char bar_label[5];
  96:   int display_bars;  /* The number of bars to display on
graphs */
  97:   int fillwidth = 0; /* Width of bars to restore when done
viewing
  98:          Units Graph */
  99:   int first_yellowbar_height = 0;
 100:   BOOL DONE;     /* User input is done */
 101:   BOOL WARNING = FALSE;
```

```
 102:   BOOL INIT_PUMP = TRUE; // initialize pump settings first
time only
 103:   BOOL IVAC_HOLD = FALSE; // true if the pump is holding

 104:   int high, low;

 105:   /***** Initialize graphics *****/
 106:   init_graphics();

 107:   /***** Title Screen *****/
 108:   title();

 109:   /***** Confirm Correct System Date & Time *****/
 110:   gmode = getgraphmode(); /* Store the current graphics mode */
 111:   restorecrtmode();   /* Switch to a text display */
 112:   datetime_check();   /* Confirm system time */

 113:   /***** Obtain Patient information + Medical Procedure +
Control Method *****/
 114:   get_patient_info();
 115:   control_type = get_control_type();

 116:   /***** Calibrate Sensor for Automatic Control Method *****/
 117:   sensorK = 1.0;  /* sensor constant is unity unless automatic
control */
 118:   FIRST_INFUSION = TRUE;
 119:   if (control_type == 2) {
 120:    get_sensor_info();
 121:    sensorK = calibrate_sensor();
 122:    process = 0; /* The initial process for automatic glucose
sensing */
 123:    labmaster_init(1); /* initialize the LabMaster 20009 */
 124:   }
 125:   else {
 126:    process = 2; /* The initial process for manual glucose
entry */
 127:   }

 128:   /***** Create Data File and Record Patient Information
*****/
 129:   create_data_file();     /* Create a data file in current
directory of C drive */
 130:   store_patient_info();   /* Write patient's personal
information to file */

 131:   setgraphmode(gmode);  /* Restore the previous graphics mode */

 132:   /***** Graphic User Interface *****/
 133:   custom_display_setup();

 134:   time(&interval_start_seconds); /* Get the current time to
satisfy initial alarm check */
 135:   time(&last_update_seconds); /* ... and initial bar line
update check */
```

```
136:    myclock = localtime(&interval_start_seconds);
137:    last_hour = myclock->tm_hour;

138:    /***** Display Patient Name As Title *****/
139:    setcolor(LIGHTCYAN);
140:    settextjustify(CENTER_TEXT, TOP_TEXT);

141:    /* Add a null terminator after the last visible */
142:    /*  character so can center it on display       */
143:    strcpy(patient_name, (char *)myfields.fields[0].data.value);
144:    for (i=strlen(patient_name); i>=0; i--) {
145:     if (isgraph((int)patient_name[i])) { /* If visible char...
*/
146:       patient_name[i+1] = '\0';    /* Place terminator after it
*/
147:       break;
148:     }
149:    }
150:    gprintf(xmax/2, border_top, patient_name);

151:    settextstyle(SMALL_FONT, HORIZ_DIR, 6);

152:    /* Display the medical procedure */
153:    setcolor(LIGHTRED);
154:    settextjustify(LEFT_TEXT, TOP_TEXT);
155:    gprintf(7, border_top, "%s", menu1_choices[procedure]);

156:    /* Display mL / Unit ratio */
157:    setcolor(YELLOW);
158:    settextjustify(RIGHT_TEXT, TOP_TEXT);
159:    gprintf(xmax-5, border_top, "1 Unit = %d mL", *((int
*)myfields.fields[4].data.value));

160:    settextstyle(SMALL_FONT,HORIZ_DIR, 8);
161:    settextjustify(LEFT_TEXT, TOP_TEXT);
162:    setcolor(WHITE);

163:    /***** Initialize Flags and Variables *****/
164:    INTERVAL_START = TRUE;  /* Beginning of time interval
between infusions */
165:    infusion_count = 0;     /* Initialize infusion counter */
166:    hour_count = 0;     /* Initialize elapsed hours */
167:    glucose_barcount = 0;  /* Initialize displayed graph bar */
168:    insulin_barcount = 0;  /*  counters to zero */

169:    /* Initialize barwidth arrays */
170:    memset(glucose_barwidth, 0, sizeof(glucose_barwidth));
171:    memset(insulin_barwidth, 0, sizeof(insulin_barwidth));
172:    memset(yellow_barwidth, 0, sizeof(yellow_barwidth));
173:    xcoord1 = xorigin1+1;
174:    xcoord2 = xorigin2+1;

175:    /***** IVAC: Set Up Serial Port *****/
176:    if (control_type==1 || control_type==2) {
```

```
177:     s_setup(PORT, COM_PARAMS);
178:    }

179:    time(&prev_second); // init prev_second for pump mode
interrogation

180:    /***** Begin insulin infusion process loop *****/
181:    /* This section governs the execution of the program and */
182:    /* facilitates a continuous flow through style. This     */
183:    /* technique allows for such things as flashing graphics */
184:    /* and string input while other processes are checked or
           updated.                                              */
185:    do {

186:     time(&current_seconds);
187:     myclock = localtime(&current_seconds);

188:     if (control_type==1 || control_type==2) {
189:      /***** IVAC: Every 9 seconds do a Mode Interrogation
190:      ******  of the pump (Mode interrogation is required
191:      ******  at least once every 20 seconds.)
192:      *****/
193:      time(&this_second);
194:      if (difftime(this_second,prev_second)>=9) {
195:       prev_second = this_second;
196:       ivac570(1, "M?", 0); // Send mode interrogation command
197:       delay(2000);
198:      }
199:     }
200:     /***** Update Screen Clock *****/
201:     current_minute = myclock->tm_min;
202:     current_hour = myclock->tm_hour;
203:     if (current_minute != last_minute) {
204:      last_minute = current_minute;
205:      clock_box();
206:      strftime(clock_str, 80, "%I:%M %p", myclock);
207:      if (strncmp(clock_str, "0", 1) == 0) /* Replace leading
zero */
208:       strnset(clock_str, ' ', 1);   /*  with space */
209:      settextstyle(SMALL_FONT, HORIZ_DIR, 0);
210:      setusercharsize(7, 5, 2, 1);
211:      settextjustify(RIGHT_TEXT, BOTTOM_TEXT);
212:      setcolor(WHITE);
213:      gprintf(xmax, border_top-5, "%s", clock_str);
214:      settextstyle(SMALL_FONT,HORIZ_DIR, 8);
215:      settextjustify(LEFT_TEXT, TOP_TEXT);

216:      /***** Every minute, calc the total amount of insulin
217:      ******  infused so far and display it
218:      *****/
219:      if (!FIRST_INFUSION) {
220:       if (procedure == 1) {/* Surgery */
221:        /* NOTE: x units/500mL * const rate of 100 mL/hr = x/5
units/hr */
```

```
 222:        insulin_units =
1.0/60.0*(float)insulin[insulin_barcount-1]/5.0;
 223:       }
 224:       else {/* Islet Transplant or Delivery */
 225:        /* NOTE: Divide by the unit quantity (usually 10mL)
 226:        ** per unit to get units
 227:        */
 228:        insulin_units =
1.0/60.0*(float)insulin[insulin_barcount-1]/((float)*((int
*)myfields.fields[4].data.value));
 229:       }

 230:       /***** Record Total Hourly Insulin Units *****/
 231:       units_per_hour[hour_count] += insulin_units;
 232:       total_insulin_units += insulin_units;

 233:       if (current_hour != last_hour) {
 234:        last_hour = current_hour;
 235:        hour_count++;
 236:       }
 237:      }

 238:      if (control_type == 2)
 239:       FIRST_INFUSION = FALSE;

 240:      /* Clear previous units message */
 241:      setfillstyle(EMPTY_FILL, 0);
 242:      bar(xorigin1, ylimit-1, xlimit2, ylimit-18);

 243:      /* Display new units message */
 244:      setcolor(LIGHTMAGENTA);
 245:      settextstyle(SMALL_FONT, HORIZ_DIR, 6);
 246:      settextjustify(CENTER_TEXT, BOTTOM_TEXT);
 247:      gprintf(xmax/2, ylimit-4, "Total insulin infused = %.1f
units", total_insulin_units);
 248:      settextstyle(SMALL_FONT,HORIZ_DIR, 8);
 249:      settextjustify(LEFT_TEXT, TOP_TEXT);
 250:      setcolor(WHITE);
 251:     }

 252:     /***** Update Bar Graphs Every 5 Seconds *****/
 253:     current_second = myclock->tm_sec;  // FOR DEMO PURPOSES ONLY
 254:     if ((difftime(current_seconds, last_update_seconds) >= 5.0)
&& !FIRST_INFUSION) {
 255:     // The following demo line updates the graph once every
second
 256:  //  if ((current_second != last_second) && !FIRST_INFUSION)
{// FOR DEMO PURPOSES ONLY
 257:  //   last_second = current_second; // FOR DEMO PURPOSES ONLY
 258:      last_update_seconds = current_seconds;
 259:      SAMPLE_SENSOR = TRUE;

 260:      /* Check if reached graphing limit on x-axis. If so        */
 261:      /*  clear graphs and redisplay last 3 bars plus the one    */
```

```
262:      /*  currently being drawn (4 in total). If there are less  */
263:      /*  than 5 bars displayed refresh the graph with fewer bars*/

264:      if (!SHOW_UNITS) {/* Don't plot if showing Units Graph */
265:       if (xcoord1 >= xlimit1) {
266:        clear_graph(xorigin1, yorigin);
267:        clear_graph(xorigin2, yorigin);

268:        /* Clear time labels */
269:        setfillstyle(EMPTY_FILL, 0);
270:        bar(xorigin1, yorigin+1, xlimit1+char_size, ymax-
3*char_size);
271:        bar(xorigin2, yorigin+1, xlimit2+char_size, ymax-
3*char_size);

272:        /***** Display the last few glucose readings and *****/
273:        /*****   infusion rates with their time labels   *****/

274:        /* Set font style for x-axis time labels */
275:        settextstyle(SMALL_FONT, VERT_DIR, 4);
276:        settextjustify(CENTER_TEXT, TOP_TEXT);

277:        xcoord1 = xorigin1+1; /* Reset plotting coords to origin */
278:        xcoord2 = xorigin2+1;

279:        if ((glucose_barcount > 1) && (insulin_barcount > 1)) {
280:         /* If more than the current bar being displayed... */
281:         display_bars = 4;
282:         if (glucose_barcount <= 6) /* If <= 6 bars displayed...*/
283:          display_bars = glucose_barcount-2;
284:         if (glucose_barcount <= 3)/* If <= 3 bars displayed...*/
285:          display_bars = glucose_barcount-1;

286:         /* Redisplay the glucose bars and time labels */
287:         for (i=0, glucose_barcount-=display_bars;
i<display_bars; i++, glucose_barcount++) {
288:          add_bar(xcoord1, yorigin,
glucose_barwidth[glucose_barcount], glucose_level[glucose_barcount],
scale1, LIGHTBLUE);

289:          /* Time labels */
290:          gprintf(xcoord1, yorigin,
&glucose_time_label[glucose_barcount][0]);

291:          xcoord1+=glucose_barwidth[glucose_barcount];
292:         }
293:         /* Redisplay the insulin bars and time labels */
294:         for (i=0, insulin_barcount-=display_bars;
i<display_bars; i++, insulin_barcount++) {
295:          add_bar(xcoord2, yorigin,
yellow_barwidth[insulin_barcount], (float)insulin[insulin_barcount-
1], scale2, YELLOW);
296:          xcoord2+=yellow_barwidth[insulin_barcount]+1;
```

```
297:          add_bar(xcoord2, yorigin,
insulin_barwidth[insulin_barcount],
(float)insulin[insulin_barcount], scale2, LIGHTRED);

298:          /* Time labels */
299:          gprintf(xcoord2, yorigin,
&insulin_time_label[insulin_barcount][0]);

300:          xcoord2+=insulin_barwidth[insulin_barcount];
301:         }

302:         xcoord1 = xorigin1+1; /* Reset plotting coords to origin */
303:         xcoord2 = xorigin2+1;

304:         /* Save the magnitude of the first yellow bar in case */
305:         /*   graph is vertically re-scaled. */
306:         first_yellowbar_height = insulin[glucose_barcount-
display_bars];

307:         /* Display bar magnitude values */
308:         for (i=0, glucose_barcount-=display_bars;
i<display_bars; i++, glucose_barcount++) {
309:          if (control_type != 2) {
310:           add_bar_label(xcoord1, yorigin,
glucose_level[glucose_barcount], scale1);

311:           xcoord2+=yellow_barwidth[glucose_barcount];
312:           add_bar_label(xcoord2, yorigin,
(float)insulin[glucose_barcount], scale2);
313:          }

314:          /* Transfer last few bar info to beginning of arrays */
315:          glucose_level[i] = glucose_level[glucose_barcount];
316:          insulin[i] = insulin[glucose_barcount];
317:          glucose_barwidth[i] =
glucose_barwidth[glucose_barcount];
318:          insulin_barwidth[i] =
insulin_barwidth[glucose_barcount];
319:          yellow_barwidth[i]  =
yellow_barwidth[glucose_barcount];
320:          strcpy(&glucose_time_label[i][0],
&glucose_time_label[glucose_barcount][0]);
321:          strcpy(&insulin_time_label[i][0],
&insulin_time_label[glucose_barcount][0]);

322:          xcoord1+=glucose_barwidth[glucose_barcount];
323:          xcoord2+=insulin_barwidth[glucose_barcount];
324:         }
325:         /* Initialize remaining portion of the array to zero */
326:         for (i=display_bars; i<=glucose_barcount; i++) {
327:          glucose_barwidth[i] = 0;
328:          insulin_barwidth[i] = 0;
329:          yellow_barwidth[i]  = 0;
330:         }
```

```
 331:         glucose_barcount = display_bars;
 332:         insulin_barcount = display_bars;

 333:         /* Return to normal font settings */
 334:         settextstyle(SMALL_FONT, HORIZ_DIR, 8);
 335:         settextjustify(LEFT_TEXT, TOP_TEXT);
 336:        }
 337:        else {/* Only a single bar is currently being displayed */
 338:         gprintf(xcoord1, yorigin,
&glucose_time_label[glucose_barcount][0]);
 339:         gprintf(xcoord2, yorigin,
&insulin_time_label[glucose_barcount][0]);
 340:         add_first_barline(xorigin1, yorigin, glucose_level[0],
scale1, LIGHTBLUE);
 341:         add_first_barline(xorigin2, yorigin, (float)insulin[0],
scale2, LIGHTRED);
 342:         glucose_barwidth[0] = 1; /* One line wide to start */
 343:         insulin_barwidth[0] = 1; /* One line wide to start */
 344:        }
 345:       }
 346:       else { /* Update the graphs with a new line */
 347:        add_barline(xcoord1, yorigin,
glucose_level[glucose_barcount-1], scale1, LIGHTBLUE);
 348:        if (process == 6) {/* i.e. waiting at 'ready' prompt */
 349:         if (insulin_barcount < 1)  /* Zero when program is first run */
 350:          add_barline(xcoord2, yorigin, 0.0, scale2, YELLOW);
 351:         else
 352:          add_barline(xcoord2, yorigin,
(float)insulin[insulin_barcount-1], scale2, YELLOW);
 353:         yellow_barwidth[insulin_barcount]++;
 354:        }
 355:        else {
 356:         if (insulin_barcount >= 1) {
 357:          add_barline(xcoord2, yorigin,
(float)insulin[insulin_barcount-1], scale2, LIGHTRED);
 358:          insulin_barwidth[insulin_barcount-1]++;
 359:         }
 360:        }
 361:       }
 362:      }
 363:      else /* Displaying the Units Graph */
 364:       fillwidth++; /* Count barlines to be restored */
 365:      glucose_barwidth[glucose_barcount-1]++;
 366:      xcoord1++;
 367:      xcoord2++;
 368:     }

 369:     /***** Check If Alarm Time Reached *****/
 370:     if (CHECK_ALARM && !NO_ALARM) {
 371:      time_difference = difftime(current_seconds,
interval_start_seconds);
 372:  //   if (time_difference >= alarm_interval*60) {
 373:      if (time_difference >= alarm_interval) {// FOR DEMO
PURPOSES ONLY
```

```
 374:       ALARM_ON = TRUE;
 375:       if (!ENTRY_LINE_CLEARED) {
 376:        entry_line_box(); /* Clear data entry line */
 377:        ENTRY_LINE_CLEARED = TRUE;
 378:        message_line_box(); /* Clear message line */
 379:       }
 380:       if (process != 9) {  /* Bypass beep while screen being
*/
 381:        alert_border(200);  /* redrawn (first beep is too long*/
 382:        beep();             /*  otherwise)        */
 383:       }
 384:     }
 385:     else
 386:      ALARM_ON = FALSE;
 387:    }

 388:    switch (process) {

 389:     case 0:  /***** Handle Alarm + Get Sensor Data *****/
 390:      /* The program will stay in process 0 unless:
 391:       *  - The alarm time limit is reached
 392:       *  - A key is pressed
 393:       *  - Input is coming from the glucose sensor
 394:       */

 395:      /***** SENSOR: Read glucose sensor voltage from LABMASTER *****/
 396:      //WITH THE CODE PLACED HERE, THE USER CAN STILL
 397:      // OVERRIDE WITH MANUAL INPUT
 398:      if (control_type==2) {
 399:       if (SAMPLE_SENSOR == TRUE) {/* Time to sample the sensor */
 400:        SAMPLE_SENSOR = FALSE;
 401:        glucose_level[glucose_barcount] = labmaster_in(1); //
use channel one

 402:  //      setcolor(0);//TEMP - for testing purposes
 403:  //      bar(10,30,65,50);//TEMP
 404:  //      setcolor(15);//TEMP
 405:  //      gprintf(10,25,"%5.1f",
glucose_level[glucose_barcount]);//TEMP

 406:        /* Convert the digital value to a voltage reading */
 407:        /* sensorK   - Sensor's constant of proportionality */
 408:        /* LM_MAXNUM - The highest digital value returned from
the LabMaster */
 409:        /* LM_MAXRNG - The maximum voltage range of the
LabMaster V or mV */
 410:        /* 1000.0    - convert mV to V */
 411:        glucose_level[glucose_barcount] =
sensorK*glucose_level[glucose_barcount]/(LM_MAXNUM/LM_MAXRNG) + 3.0
- sensorK*(*((double
*)sensorfields.fields[3].data.value))*(*((double
*)sensorfields.fields[1].data.value))/1000.0;
```

```
412:  //      glucose_level[glucose_barcount] = 3.16+rand()%5; //
FOR DEMO PURPOSES ONLY
413:        // round to 1 decimal place
414:        glucose_level[glucose_barcount] =
round(glucose_level[glucose_barcount], 1);

415:        /* Display the current glucose level */
416:        entry_line_box();  /* Clear data entry line */
417:        settextjustify(CENTER_TEXT, TOP_TEXT);
418:        gprintf(xmax/2, ymax-2*char_size-8, "Current glucose is
%.4g mmol/L", glucose_level[glucose_barcount]);
419:        ENTRY_LINE_CLEARED = FALSE;

420:        process = 5;
421:       }
422:      }

423:      /* Handle Alarm */
424:      if (!kbhit() && ALARM_ON)
425:       glucose_prompt();   /* Flash glucose prompt */
426:      else if (kbhit()) {
427:       CHECK_ALARM = FALSE;/* Pause alarm if on while
processing keyboard input */
428:       if (!ALARM_ON)
429:        process = 1; /* Check for command key press */
430:       else
431:        process = 2; /* Display normal glucose prompt */
432:      }
433:      break;

434:     case 1: /***** Handle Command Keypress *****/
435:      keypress = bioskey(0); /* Get character from keyboard
buffer */
436:      process = handle_keypress(keypress);
437:      break;

438:     case 2: /***** Display Normal Glucose Prompt *****/
439:      /* Prepare the display for the next process */
440:      if (!ENTRY_LINE_CLEARED) {
441:       entry_line_box(); /* Clear data entry line */
442:       ENTRY_LINE_CLEARED = TRUE;
443:       message_line_box(); /* Clear message line */
444:      }
445:      glucose_prompt();  /* Display glucose prompt */
446:      xinput = getx();  /* Save current coordinates */
447:      yinput = gety();  /*   for inputting text */
448:      if (getcolor() != WHITE)/* If prompt not white... */
449:       glucose_prompt(); /* Re-display glucose prompt */
450:      if (getpixel(0,ymax/2) == LIGHTRED)
451:       window_border(LIGHTCYAN);/* Reset border to original
color */

452:      process = 3;
453:      break;
```

```
454:     case 3:  /***** Get Glucose Level *****/
455:      if (!kbhit()) {
456:       if (TIMEOUT_5) {
457:        if (timeout(300)) {/* Check for input timeout > 5 min
*/
458:         CHECK_ALARM = TRUE;
459:         ALARM_ON = TRUE;
460:         TIMEOUT_5 = FALSE;
461:        }
462:       }
463:       else {
464:        if (timeout(10)) {/* Check for input timeout > 10 seconds */
465:         if (!ALARM_ON) {
466:          glucose_prompt(); /* Flash glucose prompt */
467:          delay(200);
468:         }
469:         else
470:          CHECK_ALARM = TRUE;
471:        }
472:       }
473:      }
474:      else {
475:       START_TIMEOUT = TRUE; /* Re-initialize for next usage */
476:       nosound();     /* Shut alarm off */
477:       if (getpixel(0,ymax/2) == LIGHTRED)
478:        window_border(LIGHTCYAN);/* Reset border to original
color */

479:       keypress = bioskey(0)&0xff; /* Get character from
keyboard buffer */

480:       /* Move to input coordinates. This compensates for the
*/
481:       /*  cursor being relocated when the clock is updated */
482:       moveto(xinput, yinput);

483:       /* Build glucose value string and set INTERVAL_START
484:       /*  flag when done */
485:       INTERVAL_START = get_gcfloat(keypress);

486:       xinput = getx(); /* Save current coordinates */
487:       yinput = gety(); /*  for inputting text   */

488:       if (INTERVAL_START) {
489:        if (TIMEOUT_5)
490:         TIMEOUT_5 = FALSE;
491:        INTERVAL_START = FALSE;
492:        glucose_level[glucose_barcount] = float_value;

493:        /* Display warning message if diffence between
494:        ** last two glucose levels is high */
495:        settextjustify(LEFT_TEXT, TOP_TEXT);
```

```
496:        if (fabs(glucose_level[glucose_barcount]-
glucose_level[glucose_barcount-1]) >= 10.0) {
497:         if (infusion_count > 0) {
498:          warning_message(" Large change to %.4g mmol/L",
glucose_level[glucose_barcount]);
499:          WARNING = TRUE;
500:         }
501:         else {
502:          /* Display glucose confirmation message */
503:          entry_line_box();  /* Clear data entry line */
504:          gprintf(0, ymax-2*char_size-8, "  Is glucose %.4g
mmol/L?", glucose_level[glucose_barcount]);
505:         }
506:        }
507:        else {
508:         /* Display glucose confirmation message */
509:         entry_line_box();  /* Clear data entry line */
510:         gprintf(0, ymax-2*char_size-8, "  Is glucose %.4g
mmol/L?", glucose_level[glucose_barcount]);
511:        }
512:        process = 4;
513:       }
514:       /* Cancel input if ESCape key pressed */
515:       if (keypress == ESC) {
516:        if (!ALARM_ON) { /* If alarm off simply redisplay
517:              * infusion data */
518:         display_infusion_info();
519:         process = 0;
520:        }
521:        else {
522:         /* Temporarily silence alarm (5 minutes).
523:         ** Pressing ESC a second time will cancel the
524:         ** prompt altogether and return control to
525:         ** the command line. */
526:         CHECK_ALARM = TRUE;
527:         ALARM_ON = FALSE;
528:         NO_ALARM = FALSE;
529:         TIMEOUT_5 = TRUE;
530:         time(&interval_start_seconds);
531:         process = 2;
532:        }
533:       }
534:      }
535:      break;

536:     case 4:  /***** Flash Confirmation Prompt *****/
537:      if (!kbhit()) {
538:       settextjustify(RIGHT_TEXT, TOP_TEXT);
539:       setcolor(prompt_color^8);
540:       prompt_color = getcolor();
541:       gprintf(xmax, ymax-2*char_size-8, "Confirm (Y/N) ");
542:       if (WARNING)
543:        gprintf(xmax+1, ymax-2*char_size-8, "Confirm (Y/N) ");
544:       delay(200);
```

```
545:        if (timeout(10)) { /* Check for timeout > 10 seconds */
546:         if (!ALARM_ON) {/* If the alarm is off sound a short beep */
547:          if (current_second%5 == 0)
548:           beep2();
549:         }
550:         else {
551:          alert_border(0);/* Flash border - the delay provided by */
552:          beep();         /* the flashing confirmation prompts */
553:         }
554:        }
555:       }
556:       else { /***** Confirm Entered Glucose *****/
557:        START_TIMEOUT = TRUE;
558:        nosound();
559:        if (getpixel(0,ymax/2) == LIGHTRED)
560:         window_border(LIGHTCYAN);/* Reset border to original color */
561:        setcolor(WHITE);

562:        if (WARNING) /* If warning message was displayed... */
563:         WARNING = FALSE;

564:        /* Get confirmation response */
565:        keypress = toupper(bioskey(0)&0xff);
566:        if (keypress == 'Y') {
567:         /* Display the current glucose level */
568:         entry_line_box();  /* Clear data entry line */
569:         settextjustify(CENTER_TEXT, TOP_TEXT);
570:         gprintf(xmax/2, ymax-2*char_size-8, "Current glucose is
%.4g mmol/L", glucose_level[glucose_barcount]);
571:         ENTRY_LINE_CLEARED = FALSE;
572:         INTERVAL_START = TRUE;

573:         process = 5;
574:        }
575:        else if (keypress == 'N') {
576:         settextjustify(LEFT_TEXT, TOP_TEXT);
577:         ENTRY_LINE_CLEARED = FALSE;

578:         process = 2;
579:        }
580:       }
581:       break;

582:      case 5:  /***** Begin New Glucose Bar On Graph *****/
583:       /* Re-scale if glucose level beyond vertical graphing
limit */
584:       /* The new peak becomes the scale maximum */
585:       settextstyle(SMALL_FONT, VERT_DIR, 6);
586:       total_bar_height =
glucose_level[glucose_barcount]*scale1+textwidth(gcvt(glucose_level[
glucose_barcount], 3, bar_label));
587:       if ((yorigin-total_bar_height) < ylimit) {
```

```
 588:       scale1 = (yorigin - ylimit -
textwidth(gcvt(glucose_level[glucose_barcount], 3,
bar_label)))/glucose_level[glucose_barcount];

 589:       /* Re-display glucose bars with new scale */
 590:       clear_graph(xorigin1, yorigin);/* Clear the graphing area */
 591:       xcoord1 = xorigin1+1;
 592:       for (i=0; i<=glucose_barcount; i++) {
 593:        add_bar(xcoord1, yorigin, glucose_barwidth[i],
glucose_level[i], scale1, LIGHTBLUE);
 594:        xcoord1+=glucose_barwidth[i]+1;
 595:       }
 596:       xcoord1 = xorigin1+1;
 597:       // add bar magnitude labels
 598:       for (i=0; i<=glucose_barcount; i++) {
 599:        if (control_type == 2 && !(glucose_barcount%10)) {
 600:         add_bar_label(xcoord1, yorigin, glucose_level[i],
scale1);
 601:        }
 602:        xcoord1+=glucose_barwidth[i]+1;
 603:       }
 604:       xcoord1--;
 605:      }
 606:      if (INTERVAL_START) {
 607:       /* A new interval begins with each loop for full
automatic control */
 608:       /* Begin drawing a new bar on the Glucose graph */
 609:       add_first_barline(xcoord1, yorigin,
glucose_level[glucose_barcount], scale1, LIGHTBLUE);
 610:       xcoord1++;

 611:       /***** Label current time on horizontal axis *****/
 612:       if (control_type == 2) { /* If automatic control...*/
 613:        if (!(glucose_barcount%10)) {/* ...only display time
every tenth reading from sensor */
 614:         strftime(&glucose_time_label[glucose_barcount][0], 80,
"%I:%M%p", myclock);
 615:         strlwr(&glucose_time_label[glucose_barcount][0]); /*
Convert AM/PM indicator to lower case */
 616:         if (strncmp(&glucose_time_label[glucose_barcount][0],
"0", 1) == 0)
 617:          strnset(&glucose_time_label[glucose_barcount][0], '
', 1);
 618:         settextstyle(SMALL_FONT, VERT_DIR, 4);
 619:         settextjustify(CENTER_TEXT, TOP_TEXT);
 620:         gprintf(xcoord1, yorigin,
&glucose_time_label[glucose_barcount][0]);
 621:         settextjustify(LEFT_TEXT, TOP_TEXT);
 622:         settextstyle(SMALL_FONT,HORIZ_DIR, 8);
 623:        }
 624:       }
 625:       else {
 626:        strftime(&glucose_time_label[glucose_barcount][0], 80,
"%I:%M%p", myclock);
```

```
627:         strlwr(&glucose_time_label[glucose_barcount][0]); /*
Convert AM/PM indicator to lower case */
628:         if (strncmp(&glucose_time_label[glucose_barcount][0], "0",
1) == 0)
629:          strnset(&glucose_time_label[glucose_barcount][0], ' ',
1);
630:         settextstyle(SMALL_FONT, VERT_DIR, 4);
631:         settextjustify(CENTER_TEXT, TOP_TEXT);
632:         gprintf(xcoord1, yorigin,
&glucose_time_label[glucose_barcount][0]);
633:         settextjustify(LEFT_TEXT, TOP_TEXT);
634:         settextstyle(SMALL_FONT,HORIZ_DIR, 8);
635:        }

636:        /* Calculate and display insulin infusion rate */
637:        // the insulin infusion rate is limited to a maximum of
999 mL/hr
638:        switch (procedure) {
639:         case 0:
640:          insulin[insulin_barcount] =
islet_transplant_insulin_infusion_rate(glucose_level[glucose_barcoun
t]);
641:          break;
642:         case 1:
643:          insulin[insulin_barcount] =
surgery_insulin_concentration(glucose_level[glucose_barcount]);
644:          break;
645:         case 2:
646:          insulin[insulin_barcount] =
delivery_insulin_infusion_rate(glucose_level[glucose_barcount]);
647:          break;
648:        }
649:        glucose_barcount++; /* Increment glucose bars displayed */

650:        if (FIRST_INFUSION) {
651:         FIRST_INFUSION = FALSE;
652:         start_minute = current_minute;

653:         /* Save initial start time as a string */
654:         strftime(&first_time_label[0], 80, "%I:%M%p", myclock);
655:         strlwr(&first_time_label[0]); /* Convert AM/PM
indicator to lower case */
656:         /* Strip leading 0 character from time string */
657:         if (strncmp(&first_time_label[0], "0", 1) == 0)
658:          strnset(&first_time_label[0], ' ', 1);
659:        }

660:        if (control_type == 2) /* If automatic control with
sensor input...*/
661:         process = 7;
662:        else /* If input provided by user...*/
663:         process = 6;
664:       }
665:       break;
```

```
666:     case 6:  /***** Flash Ready Prompt *****/
667:      if (!kbhit()) { /* Press key when ready to begin */
668:       settextstyle(SMALL_FONT, HORIZ_DIR, 5);
669:       settextjustify(RIGHT_TEXT, BOTTOM_TEXT);
670:       setcolor(prompt_color^8);
671:       prompt_color = getcolor();
672:       gprintf(xmax, ymax-textheight("H")-2, "Press a key ");
673:       gprintf(xmax, ymax-2, "when ready  ");
674:       delay(200);

675:       if (timeout(10)) { /* Check for timeout > 10 seconds */
676:        if (!ALARM_ON) {/* If the alarm is off sound a short beep */
677:         if (current_second%5 == 0)
678:          beep2();
679:        }
680:        else {
681:         alert_border(0);/* Flash border - the delay provided by */
682:         beep();         /* the flashing confirmation prompts */
683:        }
684:       }
685:      }
686:      else { /* ...pressed a key */
687:       START_TIMEOUT = TRUE;
688:       nosound();
689:       if (getpixel(0,ymax/2) == LIGHTRED)
690:        window_border(LIGHTCYAN);/* Reset border to original color */
691:       settextstyle(SMALL_FONT, HORIZ_DIR, 8);
692:       bioskey(0);  /* Clear keyboard buffer */

693:       process = 7;
694:      }
695:      break;

696:     case 7:  /***** Display New Infusion Rate and Update Graph *****/
697:      /***** Determine if Warning Situation *****/
698:      settextjustify(CENTER_TEXT, TOP_TEXT);
699:      if (glucose_level[glucose_barcount-1] <= 1.5)
700:       warning_message("Recommend checking or starting dextrose
IV");
701:      else if (glucose_barcount >= 2) {
702:       if ((glucose_level[glucose_barcount-2] <= 3.0) &&
(glucose_level[glucose_barcount-1] <= 3.0))
703:        warning_message("Recommend checking or starting
dextrose IV");
704:       else if ((glucose_level[glucose_barcount-2] >= 20.0) &&
705:          (glucose_level[glucose_barcount-1] >=
glucose_level[glucose_barcount-2]))
706:        warning_message("Blood glucose not falling, check
insulin IV");
707:      }

708:      if (control_type==1 || control_type==2) {
709:       /***** IVAC: Primary Mode Setup, only done first time *****/
```

```
710:        if (INIT_PUMP == TRUE) {// if first time, init pump
711:         INIT_PUMP = FALSE;
712:         // display standby message
713:         warning_message("Standby, setting up pump");

714:         if (procedure != 1) {
715:          // begin with mode interrogation to enable comm link
716:          ivac570(UNIT1, "M?", 0);
717:          delay(1500);

718:          // set initial primary rate
719:          if (insulin[insulin_barcount] <= 0) {
720:           // if the initial sample sets a rate of zero must
721:           //   change it to some value so initial setup can be
722:           //   completed. The pump will be placed on hold next time
723:           ivac570(UNIT1, "RP", 1);
724:          }
725:          else {
726:           ivac570(UNIT1, "RP", insulin[insulin_barcount]);
727:          }
728:          delay(1500);
729:          ivac570(UNIT1, "S", 0); // advance to next operating
mode
730:          delay(1500);

731:          // use prevously set primary volume-to-be-infused (VTBI)
732:          ivac570(UNIT1, "S", 0); // advance to next operating mode
733:          delay(1500);
734:          // reset total volume infused to zero

735:          ivac570(UNIT1, "VZ", 0);
736:          delay(1500);
737:          ivac570(UNIT1, "S", 0); // advance to next operating mode
738:          delay(1500);

739:          // begin primary infusion
740:          ivac570(UNIT1, "S", 0);
741:          delay(1500);
742:         }
743:         /* Display the current glucose level */
744:         entry_line_box();  /* Clear data entry line */
745:         settextjustify(CENTER_TEXT, TOP_TEXT);
746:         setcolor(WHITE);
747:         gprintf(xmax/2, ymax-2*char_size-8, "Current glucose is
%.4g mmol/L", glucose_level[glucose_barcount-1]);
748:         ENTRY_LINE_CLEARED = FALSE;
749:        }
750:        else {
751:         /***** IVAC: Change Primary Infusion Rate *****/
752:         if (procedure != 1) {

753:          // if infusion rate is zero, hold pump to avert
754:          //  an exteral alarm (EXT ALRM) condition.
755:          if (insulin[insulin_barcount] <= 0) {
```

```
 756:           if (IVAC_HOLD == FALSE) {
 757:            IVAC_HOLD = TRUE;
 758:            ivac570(UNIT1, "S", 0);
 759:            delay(1500);
 760:           }
 761:          }
 762:          // infusion rate no longer zero, run pump and set rate
 763:          else if (IVAC_HOLD == TRUE) {
 764:           IVAC_HOLD = FALSE;
 765:           ivac570(UNIT1, "S", 0);
 766:           delay(1500);
 767:           ivac570(1, "RP", insulin[insulin_barcount]);
 768:           delay(1500);
 769:          }
 770:          // set pump infusion rate to new value
 771:          else {
 772:           ivac570(1, "RP", insulin[insulin_barcount]);
 773:           delay(1500);
 774:          }
 775:         }
 776:        }
 777:       }

 778:       /* Display current insulin infusion rate */
 779:       message_line_box();  /* Clear message line */
 780:       settextjustify(CENTER_TEXT, BOTTOM_TEXT);
 781:       setcolor(WHITE);
 782:       if (insulin[insulin_barcount] > 0)
 783:        if (procedure == 1) /* Surgery */
 784:         gprintf(xmax/2, ymax-4, "INSULIN CONCENTRATION IS %d
units/500mL", insulin[insulin_barcount]);
 785:        else
 786:         gprintf(xmax/2, ymax-4, "INSULIN INFUSION RATE IS %d
mL/hr", insulin[insulin_barcount]);
 787:       else
 788:        gprintf(xmax/2, ymax-4, "INSULIN IV IS STOPPED");

 789:       /* Re-scale if infusion rate beyond vertical graphing
limit */
 790:       /* The new peak becomes the scale maximum */
 791:       settextstyle(SMALL_FONT, VERT_DIR, 6);
 792:       total_bar_height =
insulin[insulin_barcount]*scale2+textwidth(itoa(insulin[insulin_barc
ount], bar_label, 10));
 793:       if ((yorigin-total_bar_height) < ylimit) {
 794:        scale2 = (yorigin - ylimit -
textwidth(itoa(insulin[insulin_barcount], bar_label,
10)))/(float)insulin[insulin_barcount];

 795:        /* Re-display insulin bars with new scale */
 796:        clear_graph(xorigin2, yorigin);/* Clear the graphing
area */
 797:        xcoord2 = xorigin2+1;
```

```
798:        if (yellow_barwidth[0] != 0) {
799:         add_bar(xcoord2, yorigin, yellow_barwidth[0],
(float)first_yellowbar_height, scale2, YELLOW);
800:         xcoord2+=yellow_barwidth[0]+1;
801:        }
802:        for (i=1; i<=insulin_barcount; i++) {
803:         add_bar(xcoord2, yorigin, insulin_barwidth[i-1],
(float)insulin[i-1], scale2, LIGHTRED);
804:         xcoord2+=insulin_barwidth[i-1]+1;
805:         if (yellow_barwidth[i] != 0) {
806:          add_bar(xcoord2, yorigin, yellow_barwidth[i],
(float)insulin[i-1], scale2, YELLOW);
807:          xcoord2+=yellow_barwidth[i]+1;
808:         }
809:        }
810:        xcoord2 = xorigin2+1;
811:        for (i=0; i<=insulin_barcount; i++) {
812:         if (yellow_barwidth[i] != 0)
813:          xcoord2+=yellow_barwidth[i]+1;
814:         if (control_type == 2 && !(insulin_barcount%10)) {
815:          add_bar_label(xcoord2, yorigin, (float)insulin[i],
scale2);
816:         }
817:         xcoord2+=insulin_barwidth[i]+1;
818:        }
819:        xcoord2--;
820:       }

821:       if (INTERVAL_START) {
822:        add_first_barline(xcoord2, yorigin,
(float)insulin[insulin_barcount], scale2, LIGHTRED);
823:        xcoord2++;
824:       }

825:       process = 8;
826:       break;

827:      case 8:  /***** Begin New Interval *****/
828:       /* Get the time for the beginning of the interval */
829:       time(&interval_start_seconds);
830:       local_time = localtime(&interval_start_seconds);
831:       current_day = local_time->tm_mday;

832:       /***** Label new infusion start time on horizontal axis *****/
833:       strftime(&insulin_time_label[insulin_barcount][0], 80,
"%I:%M%p", local_time);
834:       strlwr(&insulin_time_label[insulin_barcount][0]); /*
Convert AM/PM indicator to lower case */
835:       if (strncmp(&insulin_time_label[insulin_barcount][0],
"0", 1) == 0)
836:        strnset(&insulin_time_label[insulin_barcount][0], ' ', 1);
837:       if (control_type == 2) { /* If automatic control...*/
838:        if (!(insulin_barcount%10)) {/* ...only display time
every tenth reading from sensor */
```

```
 839:         settextstyle(SMALL_FONT, VERT_DIR, 4);
 840:         settextjustify(CENTER_TEXT, TOP_TEXT);
 841:         gprintf(xcoord2, yorigin,
&insulin_time_label[insulin_barcount][0]);
 842:       }
 843:     }
 844:       else {
 845:        settextstyle(SMALL_FONT, VERT_DIR, 4);
 846:        settextjustify(CENTER_TEXT, TOP_TEXT);
 847:        gprintf(xcoord2, yorigin,
&insulin_time_label[insulin_barcount][0]);
 848:       }

 849:       insulin_barcount++; /* Current number of insulin bars
being displayed */

 850:       /* Construct time string for infusion start time to print
to file
 851:       ** Only display the full date if it is a new day */
 852:       if (current_day != last_day) {
 853:        strftime(time_str, 80, "%a %b %d %I:%M %p", local_time);
 854:        last_day = current_day;
 855:       }
 856:       else
 857:        strftime(time_str, 80, "          %I:%M %p",
local_time);

 858:       /***** Update patient data file *****/
 859:       /* Write time, glucose level, infusion rate/conc, and
total units to file */
 860:       if (procedure == 1) /* Surgery */
 861:        store_data("%d %s:  %g mmol/L   %s:   %d units/500mL
%.1f units",
 862:          infusion_count+1, time_str,
glucose_level[glucose_barcount-1],
insulin_time_label[insulin_barcount-1], insulin[insulin_barcount-1],
total_insulin_units);
 863:       else
 864:        store_data("%d %s:  %g mmol/L    %s:    %d mL/hr
%.1f units",
 865:          infusion_count+1, time_str,
glucose_level[glucose_barcount-1],
insulin_time_label[insulin_barcount-1], insulin[insulin_barcount-1],
total_insulin_units);

 866:       /* Clear the old time from display */
 867:       setfillstyle(EMPTY_FILL, 0);
 868:       bar(3, border_top+char_size+7, xmax-3,
border_top+char_size+17);

 869:       /* Display the new infusion start time */
 870:       setcolor(LIGHTGREEN);
 871:       settextstyle(SMALL_FONT, HORIZ_DIR, 6);
 872:       settextjustify(CENTER_TEXT, TOP_TEXT);
```

```
873:       strcpy(time_msg, "Infusion Started: ");
874:       strftime(time_str, 80, "%a %b %d %I:%M %p", local_time);
875:       strcat(time_msg, time_str);
876:       gprintf(xmax/2, border_top+char_size+2, "%s", time_msg);
877:       settextjustify(LEFT_TEXT, TOP_TEXT);
878:       settextstyle(SMALL_FONT,HORIZ_DIR, 8);
879:       setcolor(WHITE);

880:       infusion_count++; /* NOTE: Must reset this variable when
done with patient */

881:       if (control_type==2)
882:        INTERVAL_START = TRUE;
883:       else
884:        INTERVAL_START = FALSE;
885:       CHECK_ALARM = TRUE;

886:       process = 0;
887:       break;

888:      case 9:  /***** Total Insulin Units Graph Displayed *****/
889:       /* Re-display current glucose reading */
890:       if (kbhit() || ALARM_ON) {/* Restore images if key
pressed or alarm sounds */
891:        if (kbhit()) {
892:         bioskey(0);   /* Clear keyboard buffer */
893:         entry_line_box(); /* Clear data entry line */
894:         setcolor(WHITE);
895:         settextjustify(CENTER_TEXT, TOP_TEXT);
896:         gprintf(xmax/2, ymax-2*char_size-8, "Current glucose is
%.4g mmol/L", glucose_level[glucose_barcount-1]);
897:        }
898:        settextjustify(LEFT_TEXT, TOP_TEXT);

899:        /* Erase upper right hand corner beyond graph
900:        ** to account for the average value being
901:        ** displayed at the top */
902:        setfillstyle(EMPTY_FILL, 0);
903:        bar(xlimit2+1, ylimit-8, xmax-3, ylimit-1);

904:        /***** Restore Glucose & Infusion Graphs *****/
905:        putimage(2, ylimit, image_buffer1, COPY_PUT);
906:        putimage(xmax/2+1, ylimit, image_buffer2, COPY_PUT);
907:        free((void *)image_buffer1); /* Release memory */
908:        free((void *)image_buffer2); /* Release memory */

909:        /* Check to see if max horizontal graphing limit
910:        ** reached before adding in missing barwidths */
911:        if (xcoord1 < xlimit1) {

912:         /* Add in missing area on glucose and insulin
913:         ** graphs while Units Graph was displayed */
914:         add_bar(save_xcoord1, yorigin, fillwidth,
glucose_level[glucose_barcount-1], scale1, LIGHTBLUE);
```

```
 915:         add_bar(save_xcoord2, yorigin, fillwidth,
(float)insulin[insulin_barcount-1], scale2, LIGHTRED);
 916:        }
 917:        fillwidth = 0;
 918:        SHOW_UNITS = FALSE; /* No longer showing units graph */

 919:        process = 0;
 920:       }
 921:       break;

 922:      case 10:/***** Define the Value of a Unit *****/
 923:       if(kbhit()) {
 924:        keypress = bioskey(0)&0xff; /* Get character from
keyboard buffer */

 925:        /* Move to input coordinates. This compensates for the */
 926:        /*  cursor being relocated when the clock is updated */
 927:        moveto(xinput, yinput);

 928:        DONE = get_gcfloat(keypress);

 929:        xinput = getx(); /* Save current coordinates */
 930:        yinput = gety(); /*  for inputting text   */

 931:        if (DONE) {
 932:         DONE = FALSE;
 933:         *((int *)myfields.fields[4].data.value) =
(int)float_value;

 934:         display_infusion_info();

 935:         /* Erase old mL / Unit ratio */
 936:         settextstyle(SMALL_FONT, HORIZ_DIR, 6);
 937:         setfillstyle(EMPTY_FILL, 0);
 938:         bar(xmax-textwidth("--------------"), border_top+4,
xmax-3, border_top+textheight("H")+3);

 939:         /* Display mL / Unit ratio on screen */
 940:         setcolor(YELLOW);
 941:         settextjustify(RIGHT_TEXT, TOP_TEXT);
 942:         gprintf(xmax-5, border_top, "1 Unit = %d mL", *((int
*)myfields.fields[4].data.value));

 943:         settextstyle(SMALL_FONT,HORIZ_DIR, 8);
 944:         settextjustify(LEFT_TEXT, TOP_TEXT);
 945:         setcolor(WHITE);

 946:         /* Write to file */
 947:         strftime(time_str, 80, "%a %b %d %I:%M %p", myclock);
 948:         store_data("  %s:  1 Unit = %d mL", time_str, *((int
*)myfields.fields[4].data.value));
 949:         process = 0;
 950:        }
 951:        /* Cancel input if press ESCape key */
```

```
952:        if (keypress == ESC) {
953:         if (!ALARM_ON)  /* If alarm off simply redisplay info */
954:          display_infusion_info();
955:         else  /* If alarm on must enter a new glucose */
956:          CHECK_ALARM = TRUE;
957:         process = 0;
958:        }
959:       }
960:       break;

961:      case 11:  /***** Confirm Quit from Program *****/
962:       if (!kbhit()) {
963:        entry_line_box();  /* Clear data entry line */
964:        message_line_box();  /* Clear message line */
965:        settextjustify(CENTER_TEXT, TOP_TEXT);
966:        setcolor(prompt_color^8);
967:        prompt_color = getcolor();
968:        gprintf(xmax/2, ymax-2*char_size-8, "Are you sure you
want to quit? (Y/N) ");
969:        delay(200);
970:       }
971:       else {
972:        keypress = toupper(bioskey(0)&0xff);
973:        if (keypress == 'Y') { /* Quit */
974:         /* Write end time to file and record final insulin
units */
975:         strftime(time_str, 80, "%a %b %d %I:%M %p", myclock);
976:         store_data("\n  %s:  End   %.1f total units   Average =
%.1f units/hr",
977:            time_str, total_insulin_units,
minute_average(units_per_hour));
978:         process = 12;
979:         continue; /* Go to beginning of do loop */
980:        }
981:        else if (keypress == 'N') { /* Don't Quit */
982:         display_infusion_info();
983:         CHECK_ALARM = TRUE;
984:         process = 0;
985:        }
986:        /* Any other keypress returns to this process */
987:       }
988:       break;

989:      default:
990:       break;
991:     } /* End of switch statement */
992:    } while (process != 12);

993:    /***** Program Shutdown *****/
994:    graphics_cleanup();
995:    s_cleanup();   /* De-allocate resources for serial port */
996:    exit(0);
997:  }
```

```
 998:  /***** Extra Miscellaneous Functions *****/

 999:  // Round a float value to a specified number of decimal places.
1000:  float round(float real, int places)
1001:  {
1002:   int integer;

1003:   real = real*(float)pow10(places);
1004:   integer = (int)real;
1005:   real = real - integer;
1006:   if (real >= 0.5) {
1007:    ++integer;
1008:   }
1009:   real = (float)integer/pow10(places);
1010:   return(real);
1011:  }
```

BIBLIOGRAPHY

1. Statistics Canada, Health Statistics Division, Selected leading causes of death, by sex, Canada, 1995
2. J. S. Naylor, A. S. Hodel, B. Moron, D. Schumacher, "Automatic Control Issues in the Development of an Artificial Pancreas", *Proceedings of the American Control Conference*, pp. 771-775, June 1995
3. Bantam Medical Dictionary, Rev. Ed., Bantom Books, p. 120, 1990
4. W. J. Tze, "Insulin-Dependent Diabetes Mellitus, Current Concepts and Approaches", p. 274, 1991
5. "Living With Diabetes, Third Ed., Diabetic and Metabolic Centre," Edmonton General Publisher, 41, 1992
6. M B. Davidson, "Diabetes Mellitus Diagnosis and Treatment", 3rd Ed., Churchill Livingstone, p. 113, 1991
7. G. Slama, M. Hautecouverture, R. Assan, and G. Tchobroutsky, "One to five days of continuous intravenous insulin infusion on seven diabetic patients", *Diabetes*, p. 23, 1974
8. J. S. Paton, M. Wilson, J. T. Ireland, and S. B. M. Reith, "Convenient pocket insulin syringe", *Lancet*, 1, 189-190, 1981
9. J. C. Pickup, D. Rothwell, "Technology and the diabetic patient", *Medical & Biological Engineering & Computing*, 386, 1984
10. D. Rothwell, I. A. Sutherland, J. C. Pickup, J. J. Bending, H. Keen, and J. A. Parsons, "A new miniature, open-loop, extracorporeal insulin infusion pump", *J. Biomed. Eng.*, 5, 178-184, 1983

11. R. M. Froesch, K. K. Perlman, A. Bahorix, and A. M. Albisser "Clinical usefulness of short-term and long-term treatment of type I diabetics with an intravenous open-loop insulin-infuser", In *Artificial systems for insulin delivery.* Brunetti, P., Alberti, K. G. M. M., Albisser, A. M., Hepp, K. D. and M. Massi Benedetti (Eds.), Raven Press, New York, 155-160, 1983

12. P. J. Blackshear, F. D. Dorman, P. L. Blackshear, R. L. Varco, and H. Buchwald, "The design and initial testing of an Implantable infusion pump", *Surg. Gynecol. Obstet.*, 134, 51-57, 1972

13. P. J. Blackshear, "Treatment of type II patients by means of a totally Implantable insulin infusion pump", *Lancet*, 1, 1233-1235, 1981

14. G. A. Carlson, R. E. Bair, J. I. Gaona, J. T. Love, H. E. Schildbrecht, R. S. Urenda, W. J. Spencer, R. P. Eaton, and D. S. Schade, "An Implantable, remotely programmable insulin infusion system", *Med. Progr. Thr. Technology*, 9, 17-25, 1982

15. T. Shinjyo, Y. Inoue, H. Ohashi, and Y. Hirata, "Trials of insulin infusion therapy by the closed-loop and open-loop system in patients with hyperglycemia and renal insufficiency", In *Current and future therapies with insulin, Proceedings of the First International Symposium on Treatment of Diabetes Mellitus* (1983) Excerpta Medica, Amsterdam-Oxford-Princeton, 244, 241-245, 1982

16. G. F. Franklin, J. D. Powell, A. Emami-Naeini, "Feedback Control of Dynamic Systems", 3rd Ed., Addison-Wesley Pub., 1, 1994

17. M. Shichiri, R. Kawamori, Y. Yamasaki, N. Hakui, H. Abe, "Wearable Artificial Endocrine Pancreas with Needle-Type Glucose Sensor", *Lancet*, pp. 1129-1131, 1982

18. M. Moussy, D. J. Harrison, D. W. O'Brian, R. V. Rajotte, "Performance of Subcutaneously Implanted Needle-Type Glucose Sensors Employing a Novel Trilayer Coating", *Anal. Chem.*, 65, 2072-2077, 1993

19. D. A. Gough, J. C. Armour, "Development of the Implantable Glucose Sensor, What Are the Prospects and Why is It Taking So Long?", *Diabetes*, 44, 1005-1009, 1995

20. K. Camman, "Implantable Electrochemical Glucose Sensors – State of the Art", *Hormone and Metabolic Research Supplement*, 20, 4-8, 1988

21. G. Velho, Ph. Froguel, D.R. Thévenot, G. Reach, *Diabetes Nutr. Metab.*, 1988, 1, 227-234

22. G. Velho, Ph. Froguel, R. Sternberg, D.R. Thévenot, G. Reach, *Diabetes*, 1989, 38, 227-234

23. R. F. B. Turner, D. J. Harrison, R. V. Rajotte, H.P. Galtes, "A biocompatible enzyme electrode for continuous *in vivo* glucose monitoring in whole blood", *Sensors and Actuators*, B1: 561-564, 1990

24. R. F. B. Turner, D. J. Harrison, R. V. Rajotte, "Preliminary in vivo biocompatibility studies on perfluorosulfonic acid polymer membranes for biosensors applications", *Biomaterials*, 12: 361-368, 1991,

25. F. Moussy, D. J. Harrison, R. V. Rajotte, "A miniaturized Nafion-based glucose sensor: *in vitro* and *in vivo* evaluation in dogs, International Journal of Artificial Organs", Vol. 17, no. 2, 88-94, 1994

26. C. Yu, "Human Trials of an Implantable Glucose Sensor", M.Sc. Thesis, University of Alberta, 1996

27. G. G. Guilbault, G. J. Lubrano, *Anal. Chem. Acta.*, 60, 254-255, 1972

28. G. G. Guilbault, G. J. Lubrano, *Anal. Chem. Acta.*, 64, 439-455, 1973

29. L. C. Clark, C. Duggan, *Diabetes Care*, 5, 174-180, 1982

30. T. Yao, *Anal. Chem. Acta.*, 148, 27-33, 1983

31. A. V. Oppenheim, A. S. Willsky, I. T. Young, Signals and Systems, Prentice-Hall Inc., 11, 1983

32. G. Velho, Ph. Froguel, D.R. Thevenot, G. Reach, "Strategies for Calibrating a Subcutaneous Glucose Sensor", *Biomed. Biochim. Acta.*, 48, 11/12, 957-964, 1989

33. I. Sommervile, Software Engineering, 5th Ed., Addison-Wesley Pub., 420

34. Press Release, Insulin Inhaler (AERx™ System) clinical trial results announced, About.com, http://pharmacology.tqn.com/library/98news/bln0616a.htm, June, 1998

35. J. Walsh", Inhaled insulin: will it really take your breath away?", *The Diabetes Mall*, http://www.diabetesnet.com/inhale_ins.html, 1998

36. "Jet Injectors, Children with Diabetes", http://www.childrenwithdiabetes.com/d_06_350.htm, 1999

37. "Islet Transplant Procedure Moves Closer to a Cure for Diabetes", http://www.insulin-free.org/articles/heringada.htm, *Insulin-Free World Foundation*, June, 1997

38. R. Alejandro, R. Lehmann, C. Ricordi, N. S. Kenyon, M. C. Angelico, G. Burke, V. Eszuenazi, J. Nery, A. E. Betancourt, S. S. Kong, J. Miller, D. H. Mintz, "Long-Term Function (6 years) of Islet Allografts in Type 1 Diabetes", *Diabetes*, Vol. 46, 1983-1989, , 1997

39. C. Ricordi, "Human islet cell transplantation: new perspectives for an old challenge", *Diabetes Reviews*, Vol. 4, No. 3, 356-369, 1996

40. R. Walker, "Calgary researchers on road to diabetes vaccine," http://www.medicalpost.com/mdlink/english/members/medpost/data/3520/02C.HTM, Vol. 35, No. 20, 1999

41. C. Holden, "Apollo Project for the Heart?", *Science Magazine*, http://www.ibme.utoronto.ca/life/science.html, Vol. 280 12, 1681, 1998

42. E. Corcoran, "Medical Electronics", *IEEE Spectrum*, 66, Jan. 1988

43. Press Release, "MiniMed Inc. Announces Final FDA Approval for Continuous Glucose Monitoring System to Treat Diabetes", http://www.minimed.com/files/pr_35.htm, June 1999

44. V. Poitout, D. Moatti-Sirat, G. Reach, Y. Zhang, G. S. Wilson, F. Lemonnier, J. C. Klein, "A glucose monitoring system for on line estimation in man of blood glucose concentration using a miniaturized glucose sensor implanted in the subcutaneous tissue and a wearable control unit", *Diabetologia*, 36, 658-663, 1993

45. A. J. Bard, L. R. Faulkner, Electrochemical Methods, Fundamentals and Applications, John Wiley & Sons, 1980, 562-563

46. W. G. Jung, IC Op-Amp Cookbook, Third Ed., Howard W. Sams & Co., 169-171, 1986

47. G. E. Tobey, J. G. Graeme, L. P. Huelsman, Operational Amplifiers, Design and Applications, McGraw-Hill Book Co., 229-231, 1971

48. Data sheet for LM124, National Semiconductor Corp., 1-213, 1999

49. D. A. Bell, Solid State Pulse Circuits, 4th Ed., Prentice Hall Inc., 149, 1992

50. I. Sommervile, Software Engineering, 5th Ed., Addison-Wesley Pub., pp.421-422